
acm
INTRO TO
HACKATHONS & PYTHON
WORKSHOP
WITH PROF. DONNA FRENCH
VIRTUAL
1PM - 4PM
SATURDAY, 10TH OF SEPTEMBER

Introduction to Your Instructor

Donna French

Senior Lecturer & Graduate Advisor
The University of Texas at Arlington
donna.french@uta.edu

Teaching

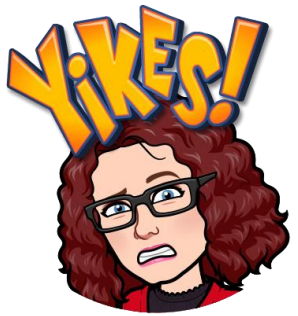
CSE 1310 – Introduction to Programming
CSE 1320 – Intermediate Programming
CSE 1325 – Object Oriented Programming
CSE 3318 – Algorithms & Data Structures

Professional Background

22 years as a Software Developer for RadioShack Corporation

Introduction to Python

Not that type of Python class!



Introduction to Python

Python is a computer programming language, a vocabulary and set of grammatical rules for instructing a computer to perform tasks.

When Guido van Rossum began implementing Python, he was also reading the published scripts from “Monty Python’s Flying Circus”, a BBC comedy series from the 1970s.

Van Rossum thought he needed a name that was short, unique, and slightly mysterious, so he decided to call the language Python.

Introduction to Python

Motto of the Python programming language

Python comes with batteries included

This stems from the fact that Python comes with a comprehensive standard library.

There are hundreds of thousands of external packages – there are supporting base libraries and packages for pretty much anything you want to accomplish.

Introduction to Python

Major Features of Python

- Enforces the use of indentation
- Python is interpreted – not compiled
- Dynamic typing
- Garbage collection

Introduction to Python

Enforces the use of indentation

Without proper indentation, your code won't run - all code blocks need to be indented to the same level.

```
def HelloWorldXY(x, y):
  if (x < 10):
    print("Hello World, x was < 10")
  elif (x < 20):
    print("Hello World, x was >= 10 but < 20")
  else:
    print("Hello World, x was >= 20")
  return x + y

for i in range(8, 25, 5):  # i=8, 13, 18, 23 (start, stop, step)
  print("--- Now running with i: {}".format(i))
  r = HelloWorldXY(i,i)
  print("Result from HelloWorld: {}".format(r))
```

Introduction to Python

Python is interpreted - not compiled

Code is written in a text editor and executed directly – no additional steps on compiling or linking.

Code is platform independent – as long as a platform has a Python interpreter, code will run.

Interpreted languages are generally not high performance.

Introduction to Python

Dynamic Typing

Java

```
String Name = "Python";
int age = 30;
double version = 3.9;
```

C

```
char Name[10] = "Python";
int age = 30;
double version = 3.9;
```

Pro

Can make it easier to get started quickly

Con

Can be more error prone

Python

```
Name = "Python"
age = 30
version = 3.9
```

Strongly typed languages won't compile when a type error is made, but Python would with unexpected results.

Introduction to Python

Garbage Collection

Recycle that used memory!

Variables take up space in memory.

The amount of available memory is limited.

In order to not run out, some languages require the programmer to clean up unused variables.

If some cleanup is missed, then memory leaks can occur and the program can slow or crash.

Python has a garbage collector that handles this cleanup for you.

Introduction to Python

We are going to write our Python code using

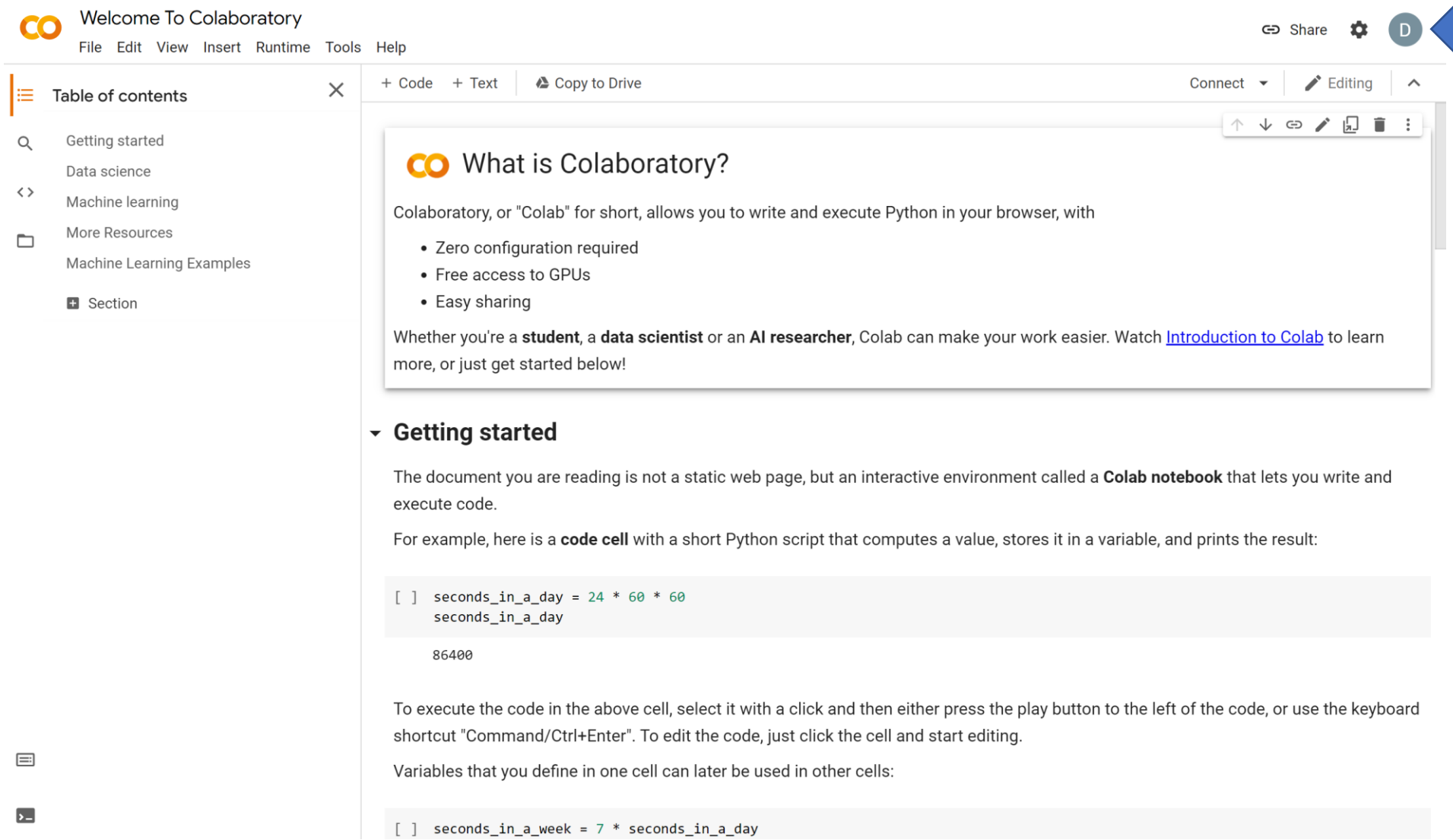
Colaboratory from Google Research

colab.research.google.com

Colaboratory, or "**Colab**" for short, allows you to write and execute Python in your browser with

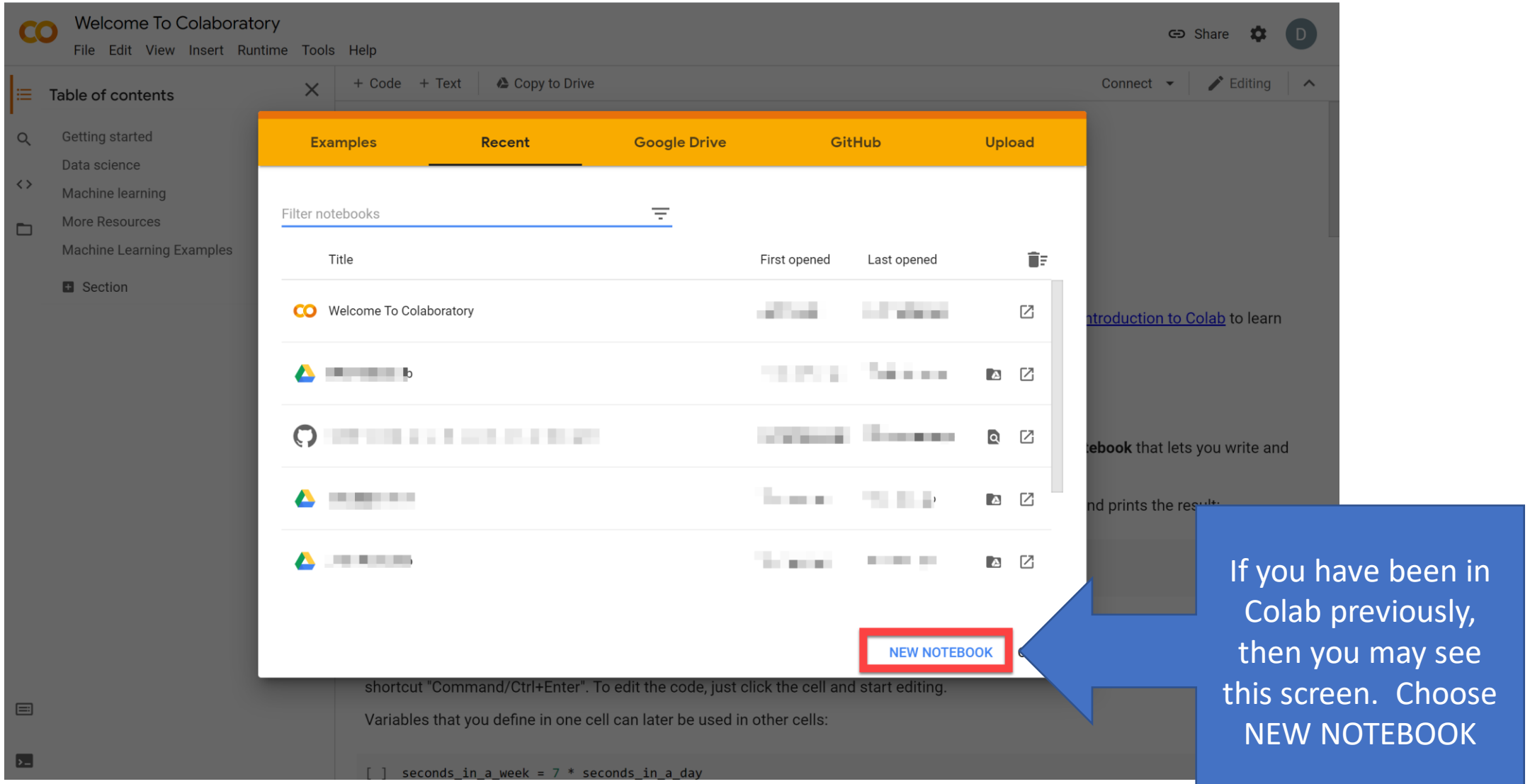
- Zero configuration required
- Free access to GPUs
- Easy sharing

Introduction to Python

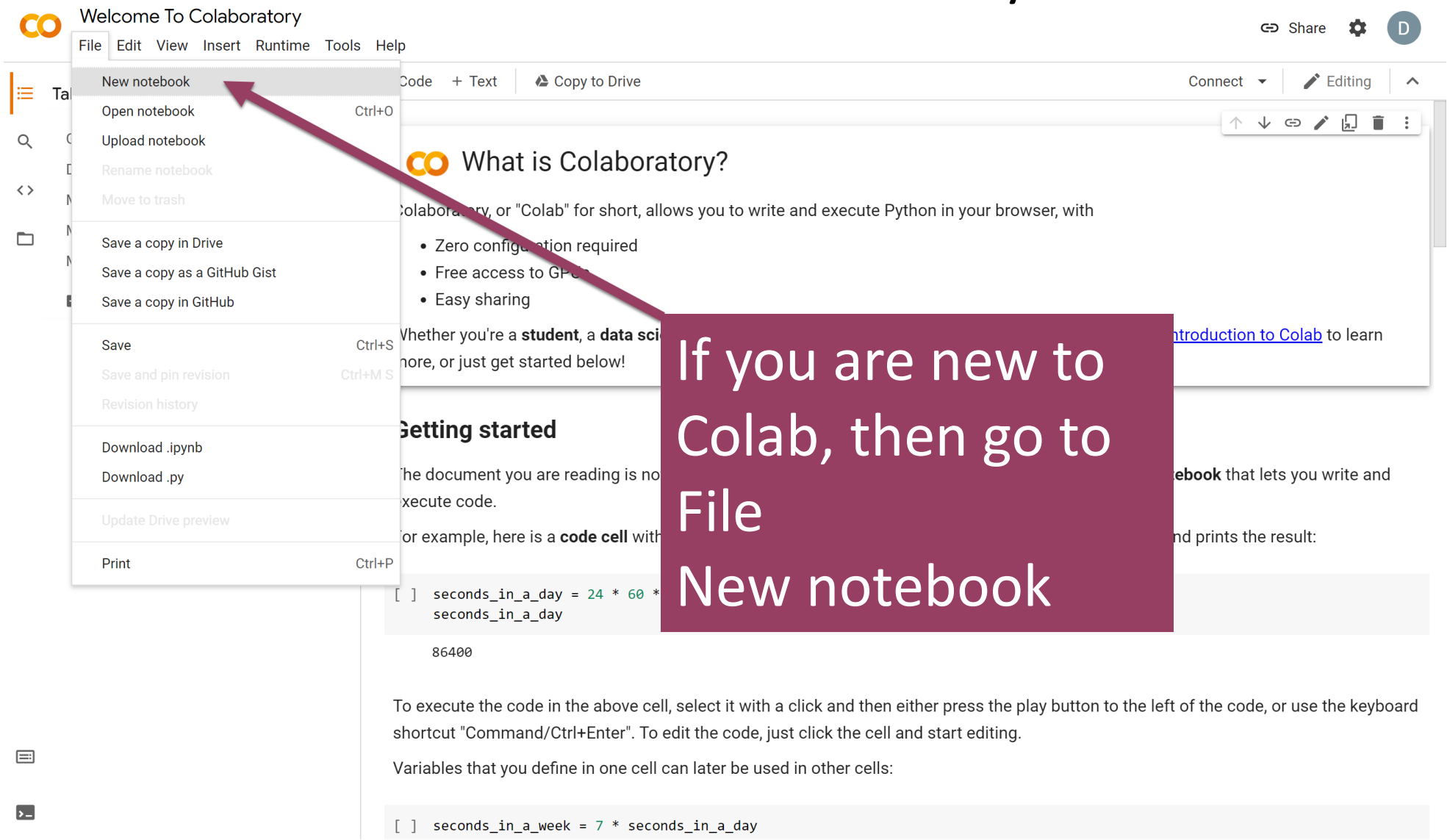


Sign in with your Google account.

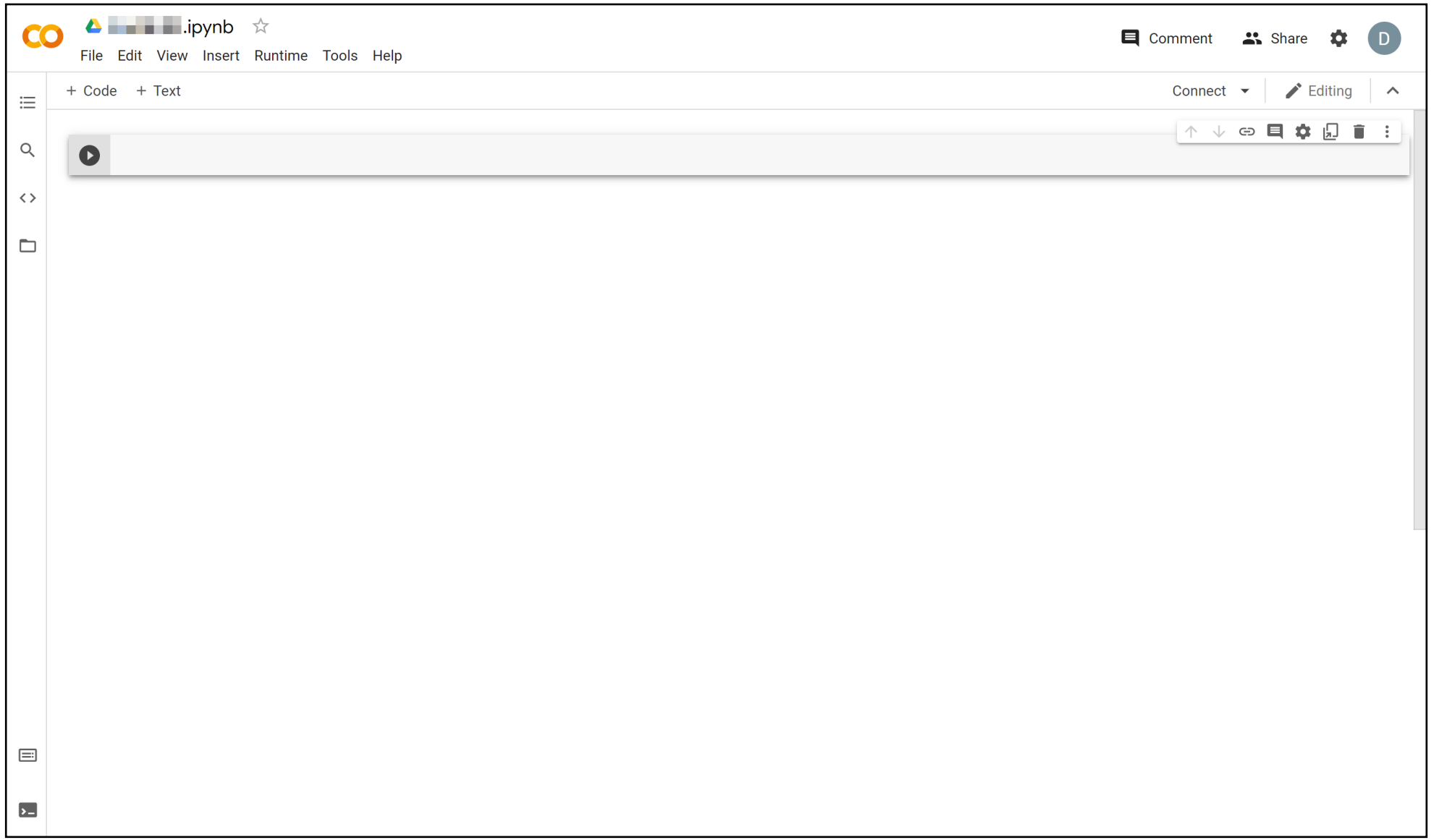
Introduction to Python



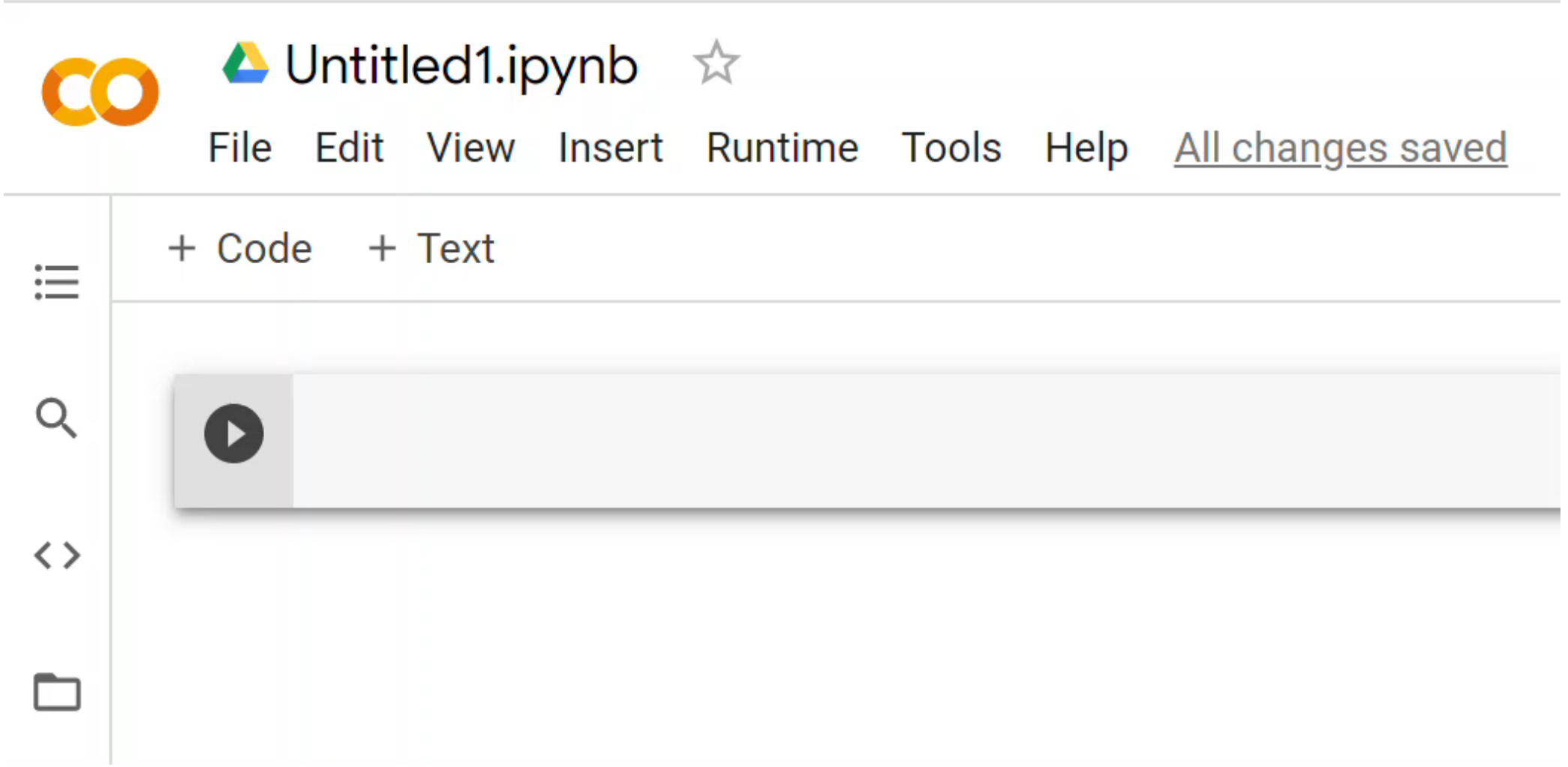
Introduction to Python



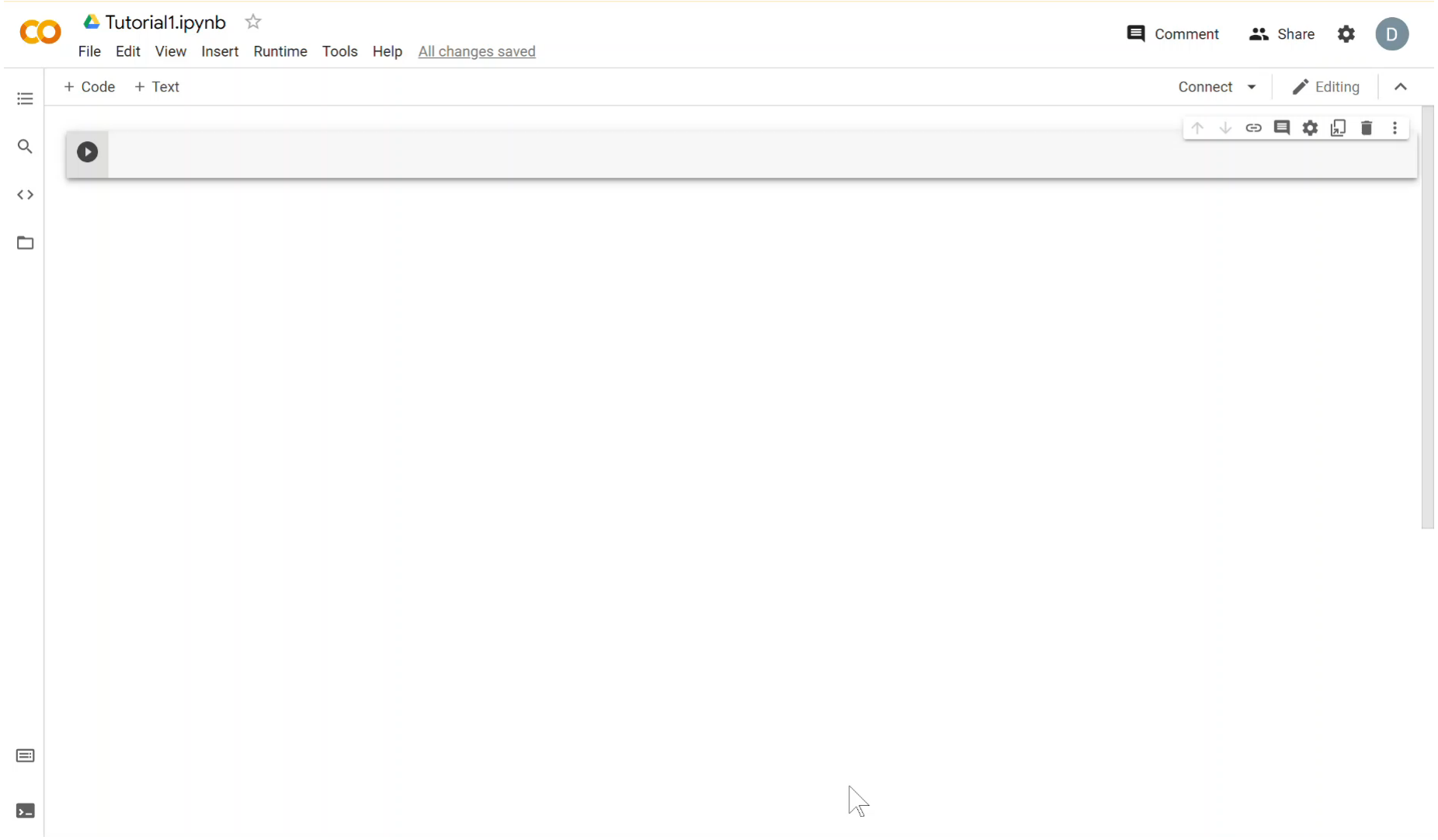
Introduction to Python



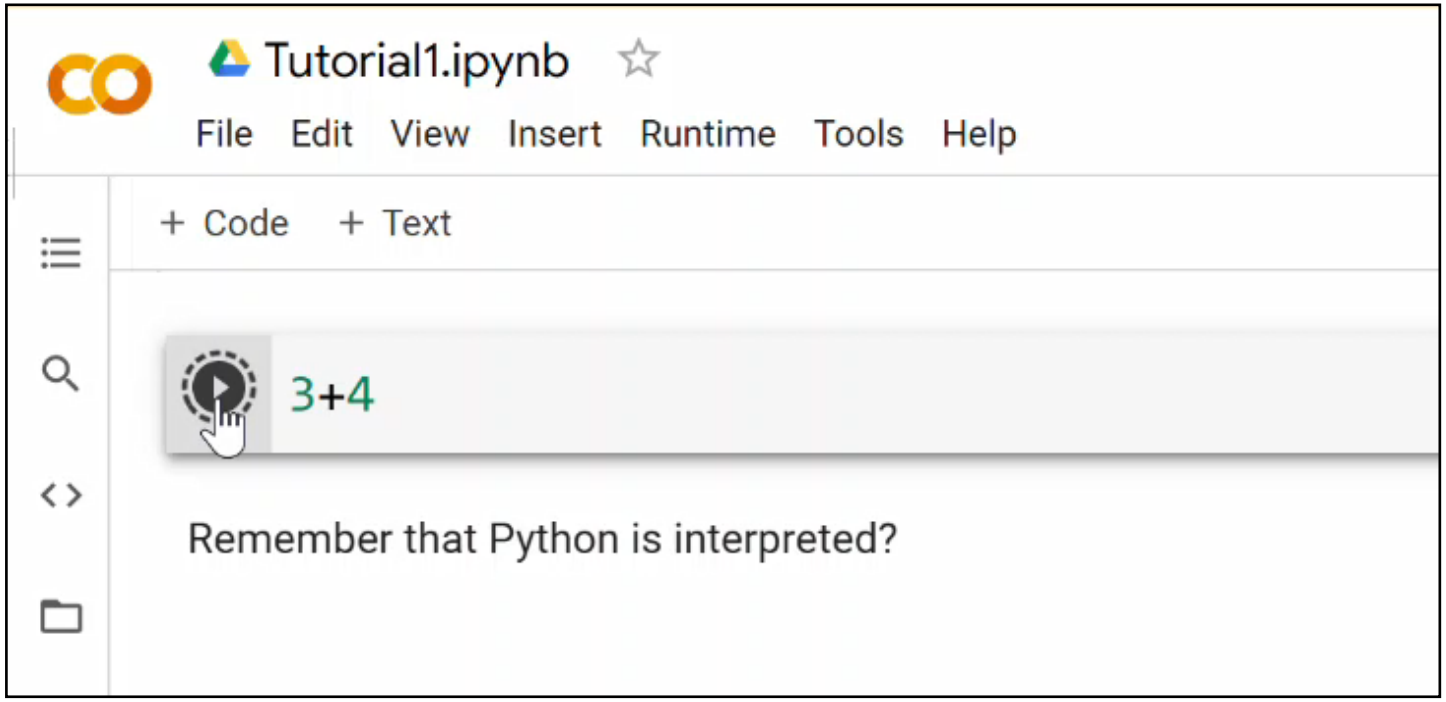
Introduction to Python



Introduction to Python



Introduction to Python



To execute a line of code, the mouse can be used to click on the

icon.

The keyboard shortcut of

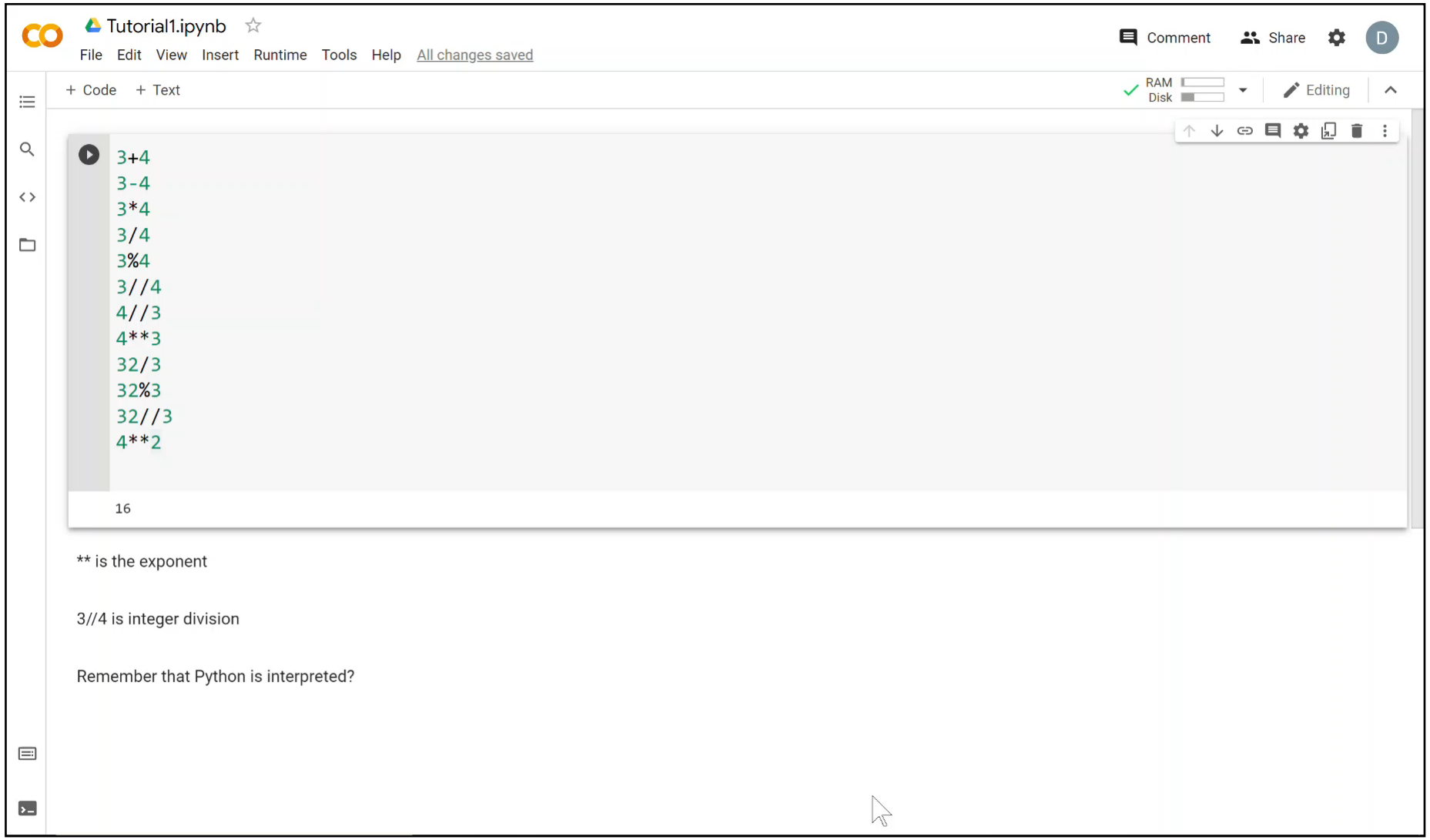
Ctrl-ENTER

will also execute the code.

Introduction to Python

Operator	Description	Example	Result
+	Addition	3 + 4	7
-	Subtraction	7 – 2	5
*	Multiplication	5 * 6	30
/	Division	32 / 3	10.666666666666666
%	Modulus (mod)	32 % 3	2
//	Integer Division	32 // 3	10
**	Exponential	4 ** 2	16

Introduction to Python



To undo

Ctrl-M-Z

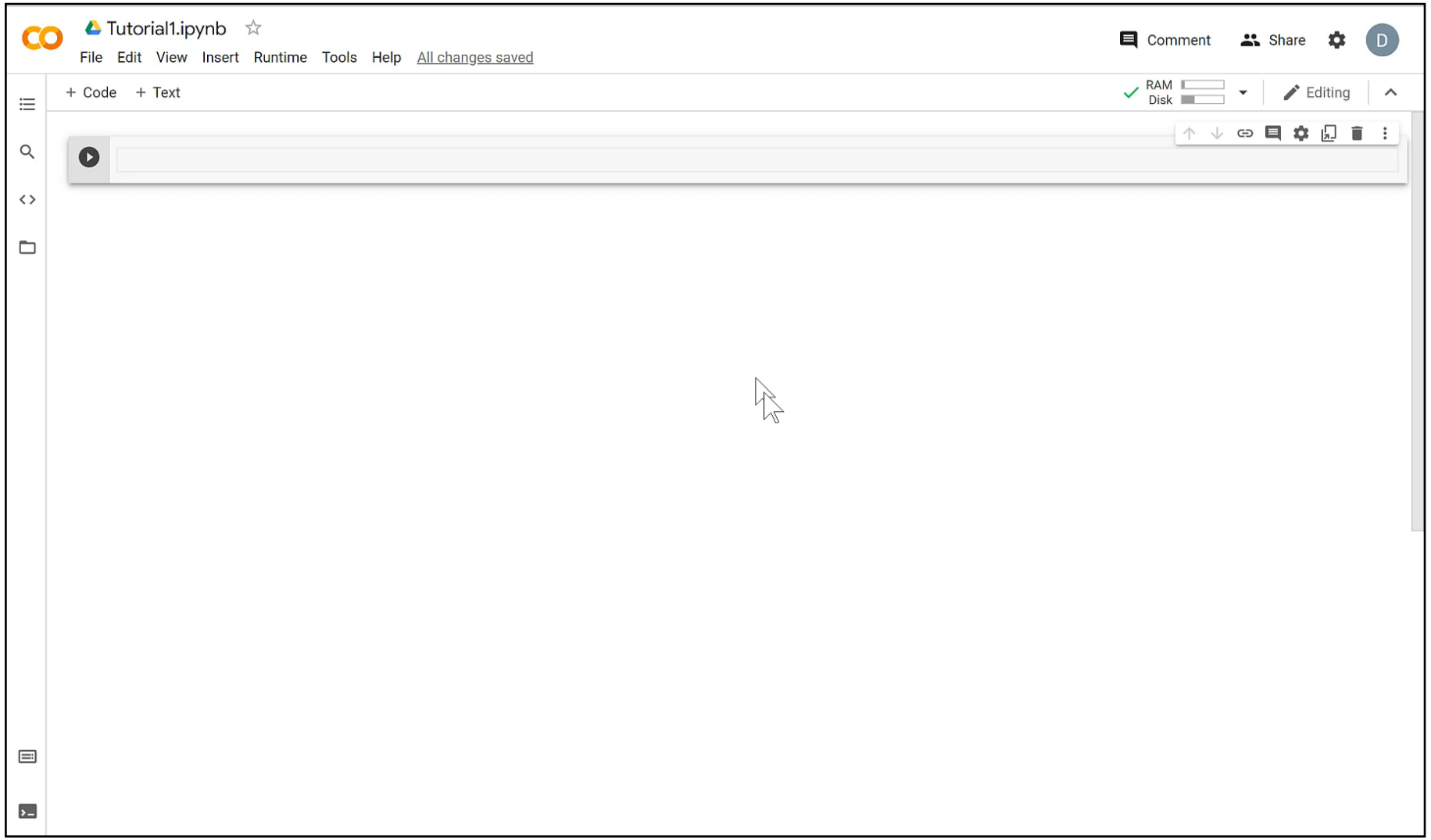
To delete

Ctrl-M-D

To insert a new code cell

Ctrl-M-B

Introduction to Python

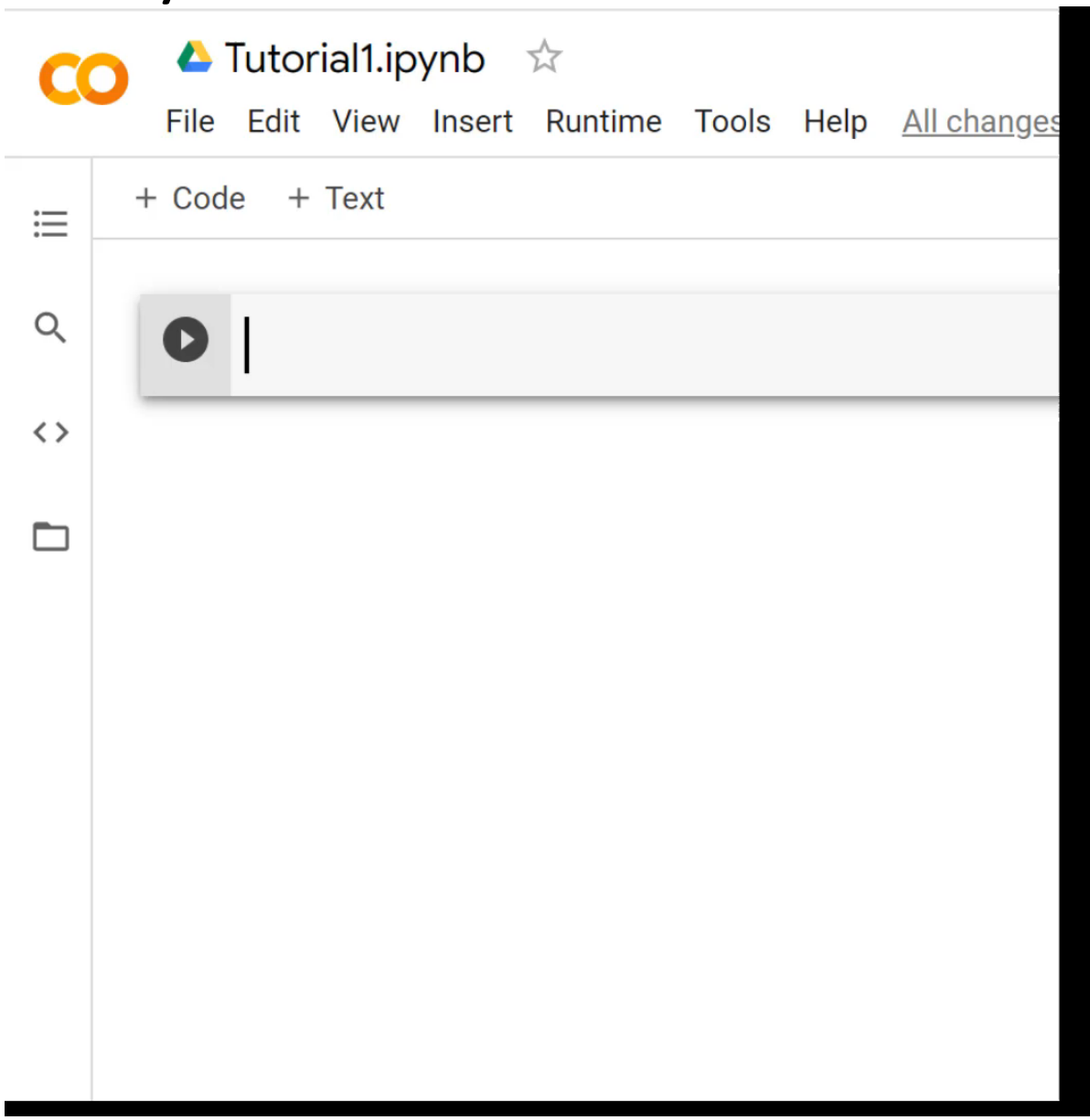


Naming Variables in Python

Introduction to Python

Did you notice in the previous example that we did not state a type for the variable `Dog`?

That is because Python uses dynamic typing - the Python interpreter does type checking only as code runs and the type of a variable is allowed to change over its lifetime.



Introduction to Python

Dynamic typing can lead to some unexpected results if you are not careful…

or

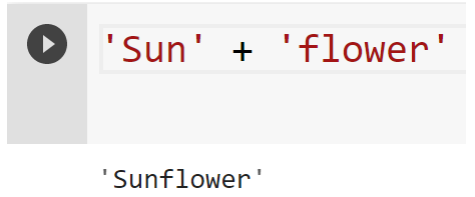
you can take advantage of it….

Introduction to Python

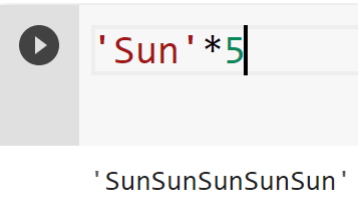
Strings

A string is a sequence of characters.

Using + with two strings will concatenate them together.



Using * with a string and a value repeats the string



Introduction to Python

Strings

- and / are not defined for strings

```
'Sunflower' - 'flower'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-13-ca138a268067> in <module>()
----> 1 'Sunflower' - 'flower'

TypeError: unsupported operand type(s) for -: 'str' and 'str'
```

```
'Sunflower' / 'flower'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-14-d0cb6d15cbd7> in <module>()
----> 1 'Sunflower' / 'flower'

TypeError: unsupported operand type(s) for /: 'str' and 'str'
```

```
'Sunflower' / 2
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-18-110778b1cb6d> in <module>()
----> 1 'Sunflower' / 2

TypeError: unsupported operand type(s) for /: 'str' and 'int'
```

Introduction to Python

Strings

So…single quotes or double quotes?

```
"Sun" + "flower"
```

```
'Sunflower'
```

```
'Sun' + 'flower'
```

```
'Sunflower'
```

' and " are interchangeable *most* of the time…

Introduction to Python

Strings

```
'A Star Wars quote is "May the Force be with you"'
```

```
'A Star Wars quote is "May the Force be with you"'
```

```
"A Star Wars quote is "May the Force be with you""
```

```
  File "<ipython-input-26-2e6128d72519>", line 1
    "A Star Wars quote is "May the Force be with you""
                          ^
SyntaxError: invalid syntax
```

```
"A Star Wars quote is \"May the Force be with you\""
```

```
'A Star Wars quote is "May the Force be with you"'
```

Introduction to Python

Strings

```
'How's it going?''
```

```
  File "<ipython-input-46-7e4065b946b0>", line 1
    'How's it going?''
         ^
SyntaxError: invalid syntax
```

```
'How\'s it going?'
```

```
'How's it going?'
```

```
"How's it going?"
```

```
'How's it going?'
```

Use " instead of ' to quote the string

Use \ to escape the '

Introduction to Python

Strings

We can calculate the length of a string by using `len()`

```
len("supercalifragilisticexpialidocious")
```

```
34
```

```
len("supercalifragilisticexpialidocious")*5
len("supercalifragilisticexpialidocious"*5)
```

```
170
```

Introduction to Python

Strings

`len()` expects a string

```
Cat = 3
len(Cat)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-39-5644914a8361> in <module>()
      1 Cat = 3
----> 2 len(Cat)

TypeError: object of type 'int' has no len()
```

```
Cat = 'Shade'
len(Cat)
```

```
5
```

```
Cat = 3.1
len(Cat)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-41-cd8c5a59a543> in <module>()
      1 Cat = 3.1
----> 2 len(Cat)

TypeError: object of type 'float' has no len()
```

Introduction to Python

Built-in Functions

```
Color = 'Red'
print(Color)
```

```
Red
```

How do we print something to the screen?

The built-function print takes either a string variable or a quoted string and prints it.

```
print("Purple")
```

```
Purple
```

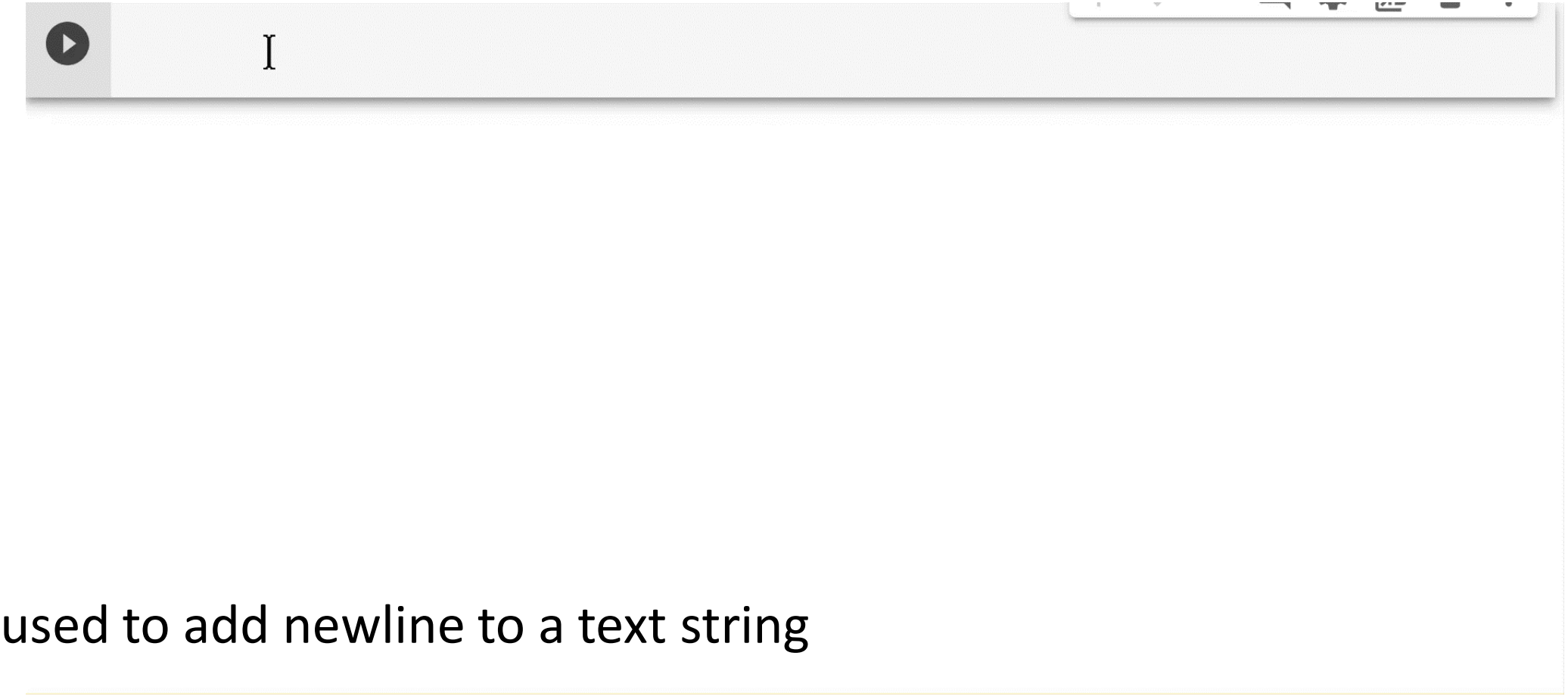
```
Color1 = 'Red'
Color2 = 'Blue'
print(Color + Color2)
```

```
RedBlue
```

```
print("Purple"*5)
```

```
PurplePurplePurplePurplePurple
```

Introduction to Python



`\n` is used to add newline to a text string

Use multiline continuation character `\` to break a statement into multiple lines.

Introduction to Python

How do we format our output? We use the built-in function

`format`

`format` is used by substituting replacement fields for placeholders in the string using the formats specified within the placeholder.

`format(replacementField, …)`

Introduction to Python

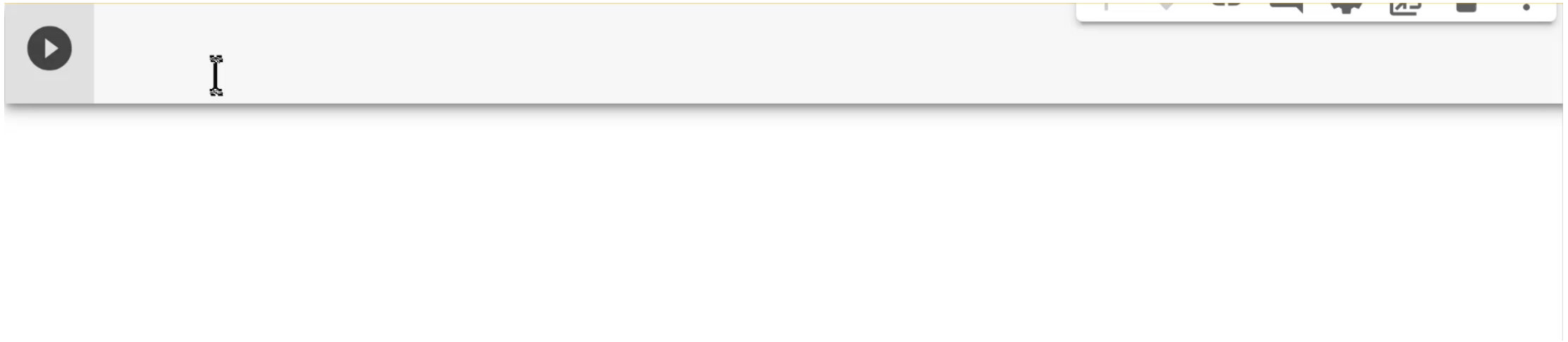
How do we format our output? We use the built-in function

`format`

`format` is used by substituting replacement fields for placeholders in the string using the formats specified within the placeholder.

`format(replacementField, …)`

Introduction to Python



Introduction to Python

Type of Replacement Field	Type Conversion Character	Output Format
String	`s`	String in default format
Integer	`c`	Converts an integer to its Unicode equivalent
	`d`	decimal integer (default)
Floating point	`f or F`	Floating point with a precision of 6
Precision	`.n`	Display `n` numbers after the decimal point

```
x = "Hi"
y = 33
z = 1
a = 2.3
b = 3.14156
print("{0:s}{1:c} {2:d} {3:f} {4:.2f}".format(x,y,z,a,b))
```

Introduction to Python

Built-in Functions

We just learned how to use `len()` to get the length of a string.
Function `len()` is built into the language.

Some built-in functions return values and other do not

```
ReturnValue = print("Purple"*5)
print(ReturnValue)
```

```
PurplePurplePurplePurplePurple
None
```

None is a special value that means nothing was returned

```
ReturnValue = len("Purple")
print(ReturnValue)
```

```
6
```

Introduction to Python

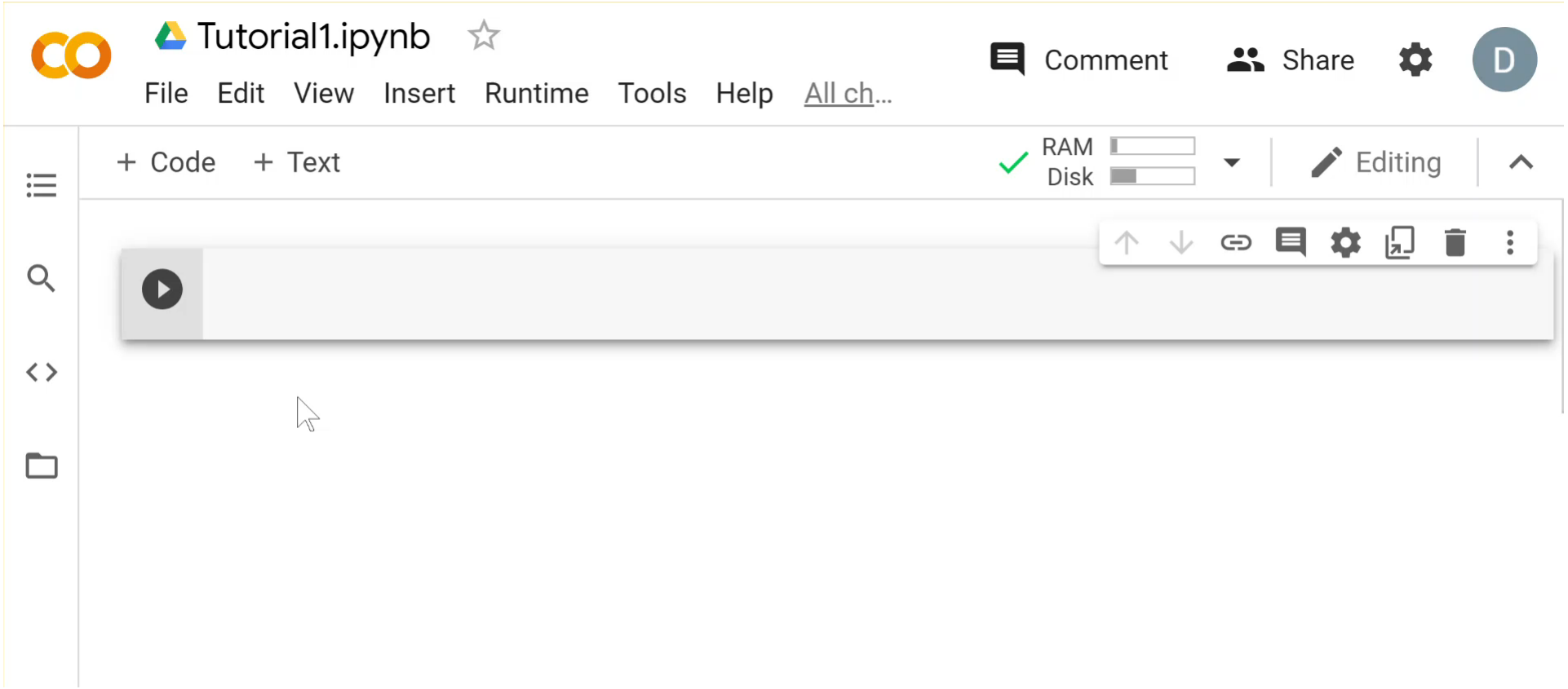
Let's create our own functions!

Write a function name `Hello` that prints "`Hi!`" to the screen.

Does not take any parameters and does not return anything.

Call function `Hello`

Introduction to Python



Introduction to Python

```
def Hello():
    print("Hi!")
Hello()
```

`def` is the Python keyword that tells it we want a function

`Hello` is our function's name

The `()` are where we will put function arguments when we need to

empty `()` means we are not passing anything to this function

`:` signals the end of the line of the function definition

Introduction to Python

```
def Hello():
    print("Hi!")
Hello()
```

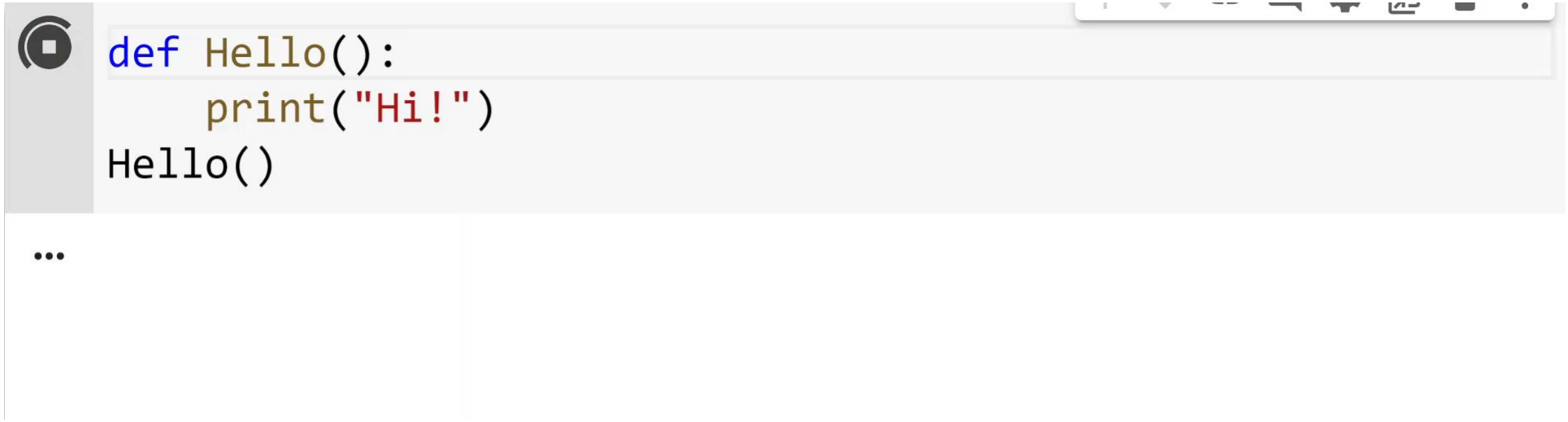
`print("Hi!")` is the first (and only) line of our function.

Notice that it is indented.

The next line is not.

`Hello()` calls our new function.

Introduction to Python



Colab uses a standard indent of 4 spaces. Using the TAB will get that indentation

Best Practice – Use TAB

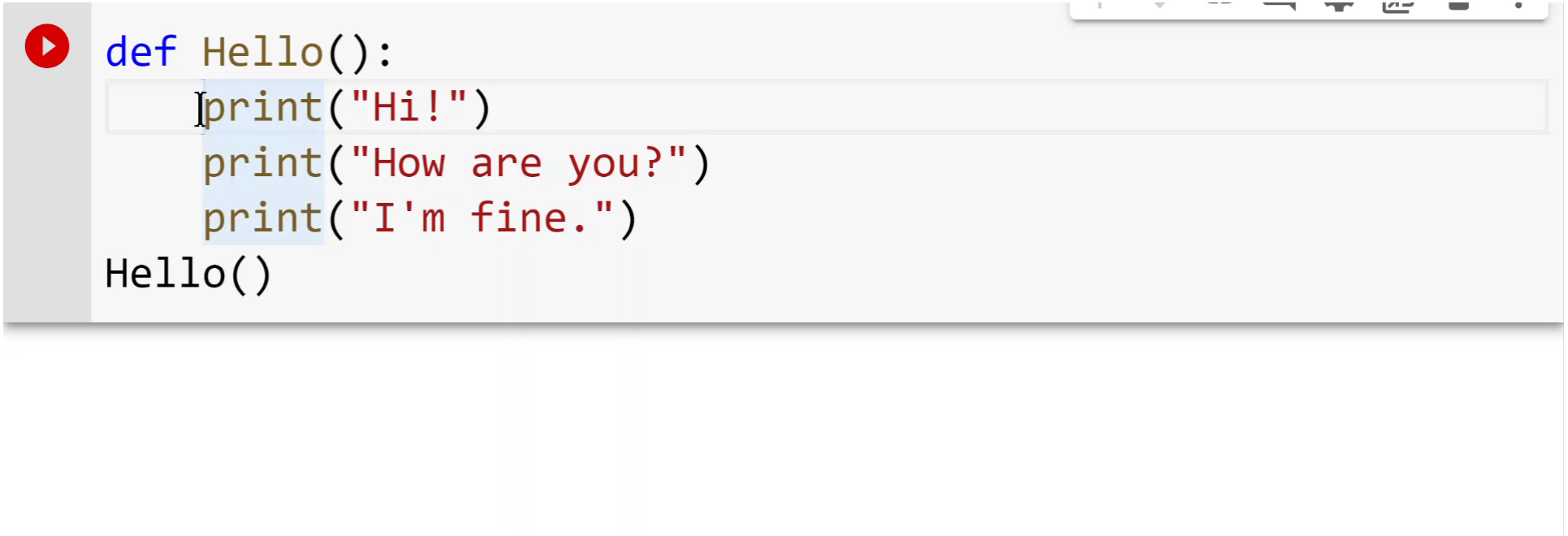
Introduction to Python

Let's add another line to our function

```
def Hello():
    print("Hi!")
Hello()
```

Introduction to Python

Consistent indentation is required.



What happened? Why did "I'm fine" print before "Hi!"

Introduction to Python

Let's add another function.

```
def Hello():
    print("Hi!")
    print("How are you?")
    print("I'm fine!!")
Hello()
```

Introduction to Python

```
def Hello(name):
    print("Hi", name, "!")
    print("How are you?")
    print("I'm fine.")
Hello("Python")
```

```
Hi Python !
How are you?
I'm fine.
```

A variable containing the string can be passed to the function.

Introduction to Python

```
def Hello(name):
    print("Hi", name, "!")
    print("How are you?")
    print("I'm fine.")
language = 'Python'

Hello(language)
```

```
Hi Python !
How are you?
I'm fine.
```

More than one parameter can be passed

Introduction to Python

```
def WhatDay(Date, DOW):
    print("Today is", DOW, Date)
WhatDay('02-13-2021', 'Saturday')
```

Parameters can be named.

Introduction to Python

Named parameters

```
def WhatDay(Date, DOW):
    print("Today is", DOW, Date)
WhatDay(DOW='Saturday',Date='02-13-2021')
```

The name of the parameter in the function call must match the name of the parameter in the function's signature.

This allows the function call to list the parameters in any order.

Introduction to Python

Parameters can be given default values in the function's signature

```
def WhatDay(Date, DOW):
    print("Today is", DOW, Date)
WhatDay(DOW='Saturday',Date='02-13-2021')
```

```
Today is Saturday 02-13-2021
```

Introduction to Python

Be cautious using default values

```
def WhatDay(Date='02-14-2021', DOW='Sunday'):
    print("Today is", DOW, Date)
WhatDay()
```

```
Today is Sunday 02-14-2021
```

Introduction to Python

```
def WhatDay(Date='02-14-2021', DOW='Sunday'):
    print("Today is", DOW, Date)
WhatDay('Saturday')
```

Default parameters are assigned from left to right.

So whatever is passed first gets assigned to the first parameter of the function.

'Saturday' was assigned to Date and not to DOW.

Introduction to Python

This ordering issue can be avoiding by using named parameters with default values.

```
def WhatDay(Date='02-14-2021', DOW='Sunday'):
    print("Today is", DOW, Date)
WhatDay('Saturday')
```

```
Today is Sunday Saturday
```

Introduction to Python

How do functions in Python return values?

```
def PickALanguage():
    print('1. C')
    print('2. Java')
    print('3. Python')
    print('4. C++')
    Choice = input("Pick your favorite ")
    print("You picked {0:s}?!?".format(Choice))

PickALanguage()
```

Introduction to Python

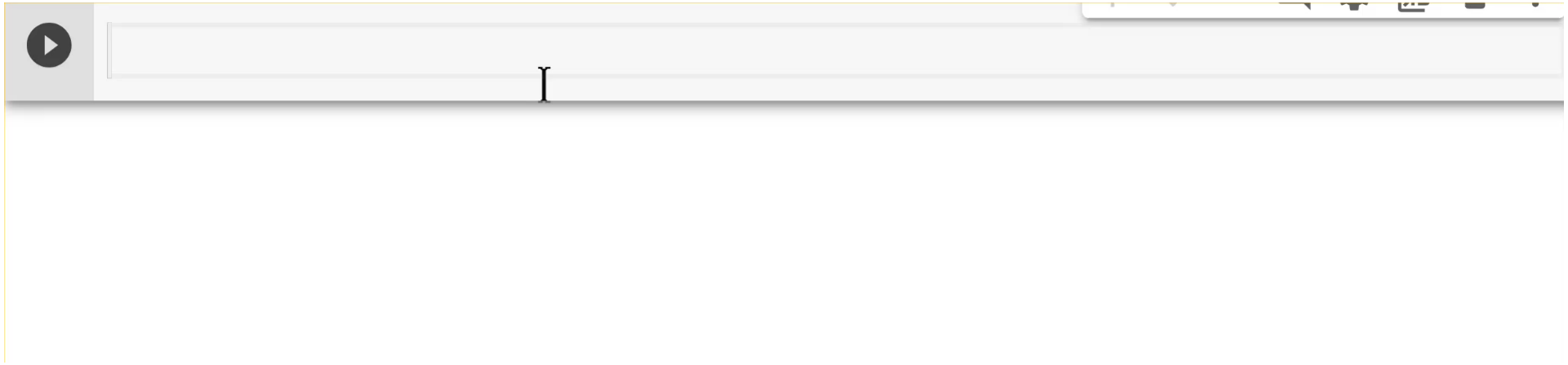
How do functions in Python return values?

```
def Multiple(a, b):
    return a*b

Multiplicand = float(input("Enter first value "))
Multiplier = float(input("Enter second value "))
result = Multiple(Multiplicand, Multiplier)
print("The result is {0:.3f}".format(result))
```

Introduction to Python

Python uses the Boolean values of

`True` `False`



Introduction to Python

So how do we prompt for input in Python?

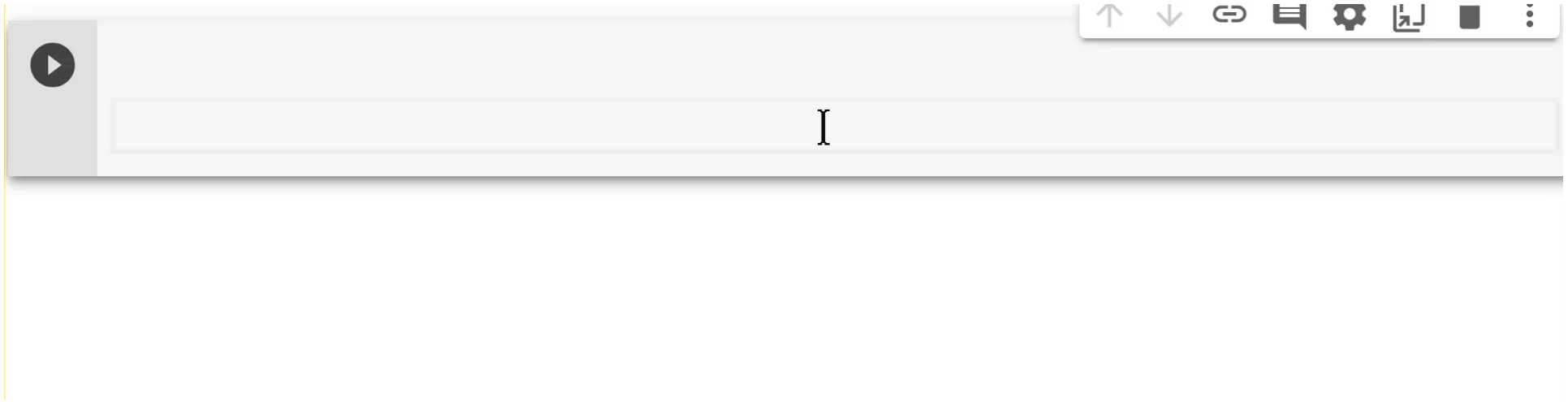
Let's look at the built-in function `input`

Format for using `input`

```
result = input(prompt)
```

`result` will be a string and **`prompt`** is typically a quoted string

Introduction to Python



Note that input does not put a space after the end of the prompt.

Introduction to Python

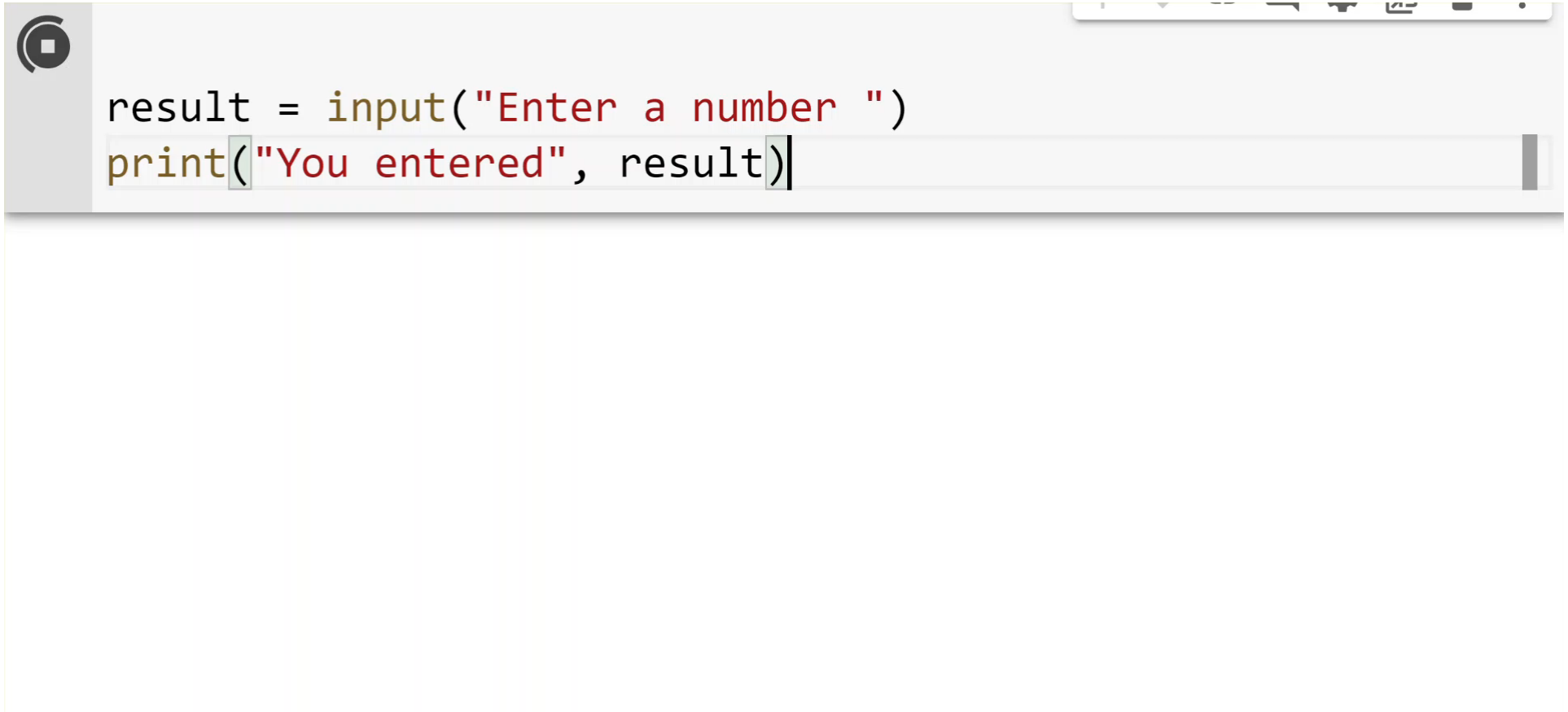
```
result = input(prompt)
```

`result` will be a string and `prompt` is typically a quoted string

Built-in functions can be used to convert the string result of `input` to other data types

For example, the return string can be converted to `int` or `float`.

Introduction to Python



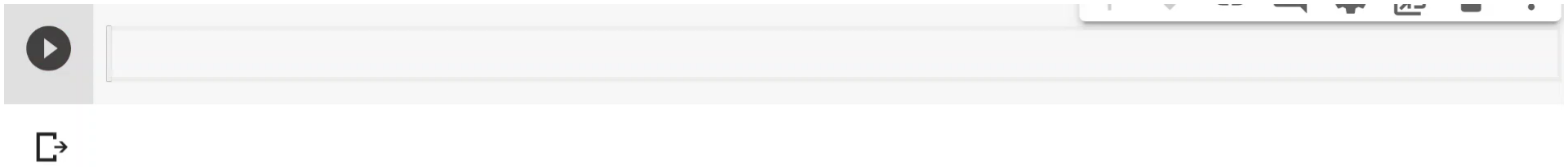
`(int)result` and `(float)result`

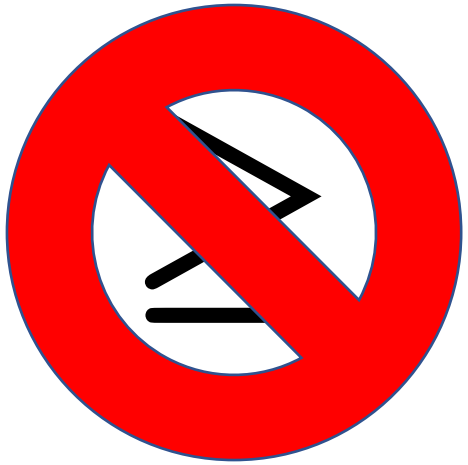
Introduction to Python

Python uses the Boolean values of

`True` has a value of `1`

`False` has a value of `0`





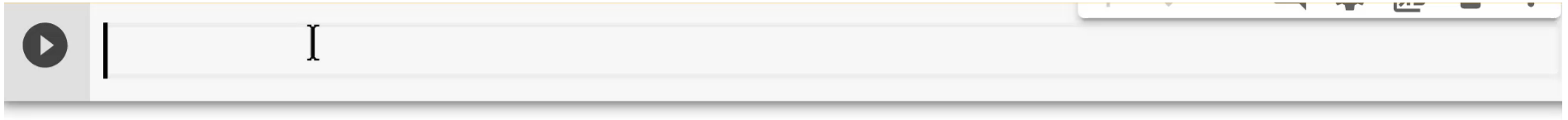
Introduction to Python

Relational Operators in Python

Operator	Meaning
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
==	is equivalent to
!=	is NOT equivalent to

Introduction to Python

Relational Operators in Python



ASCII

Introduction to Python

Logical Operators in Python

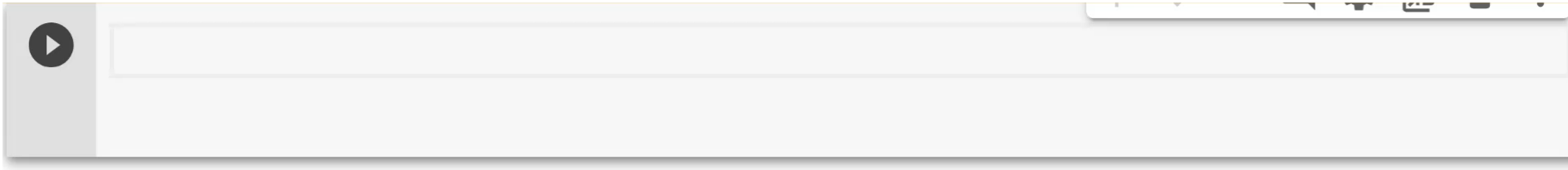
`and or not`

Logical Operator	Behavior

```
0   and 1
0
1   and 0
0
0   and -1
0
-1  and 0
0
1   and 1
1
3   and 9
9
-5  and -10
-10
```

Introduction to Python

Logical Operator	Behavior
`x and y`	if `x` is `False`, then return `x` if `x` is `True`, then return `y`



```
0   or 1
1
1   or 0
1
0   or -1
-1
-1  or 0
-1
1   or 1
1
3   or 9
3
-5  or -10
-5
```

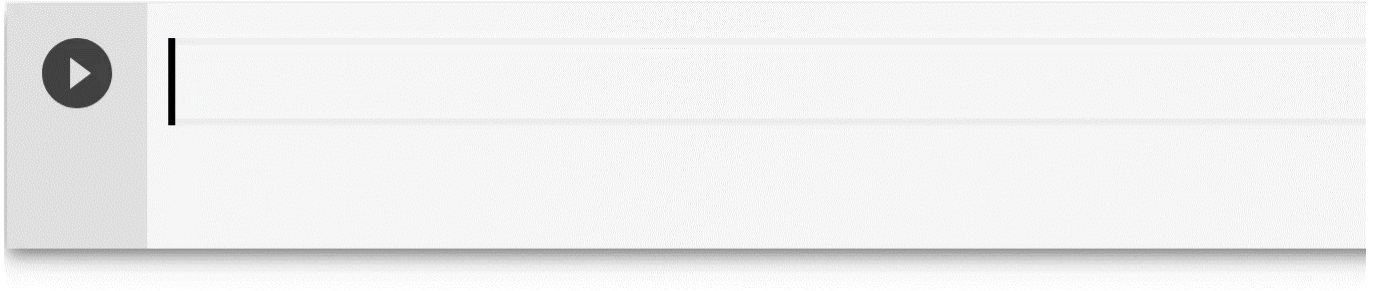
Introduction to Python

Logical Operator	Behavior
`x or y`	if `x` is `False`, then return `y` if `x` if `True`, then return `x`

```
not -1
False

not 0
True

not 1
False

not 123456789
False

not -123456789
False
```

Introduction to Python

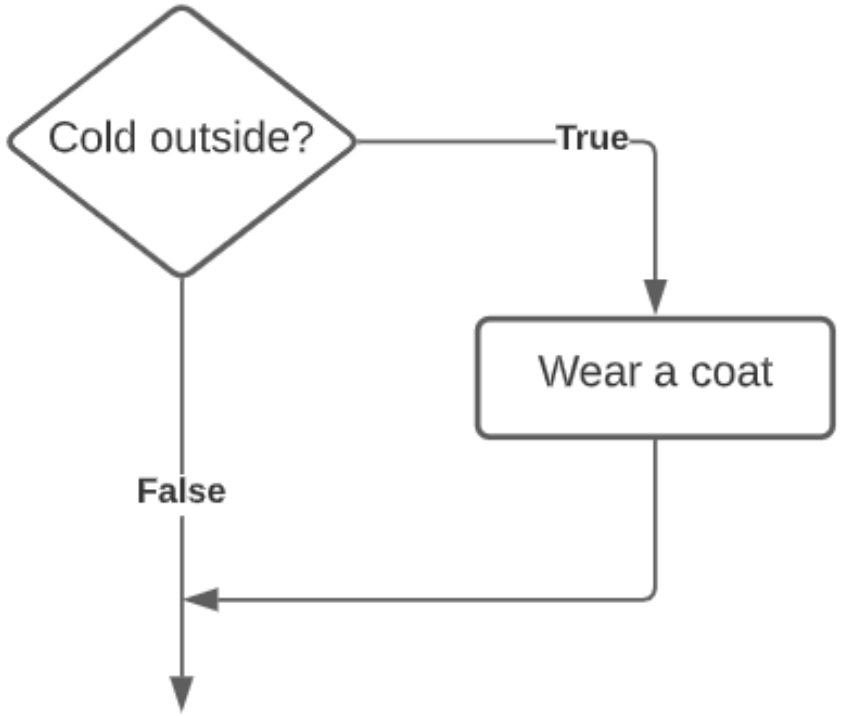
Logical Operator	Behavior
`not x`	if `x` is `False`, then return `True` if `x` is `True`, then return `False`



Introduction to Python

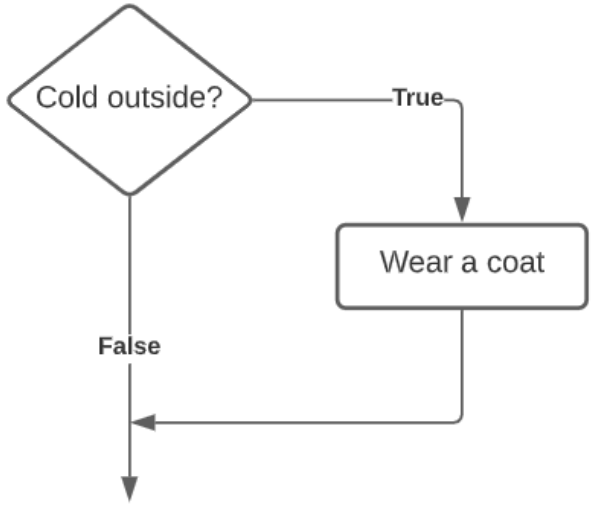
Conditional Statements

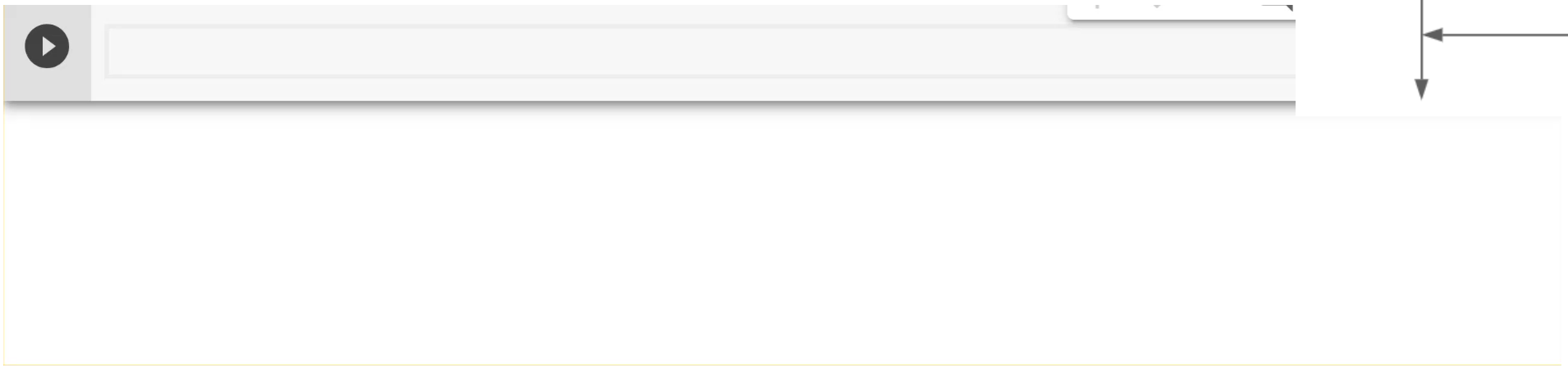
```
if condition:
    statements
```



Introduction to Python

Conditional Statements





( ) around the condition are unnecessary but will compile. Just get the : after the)

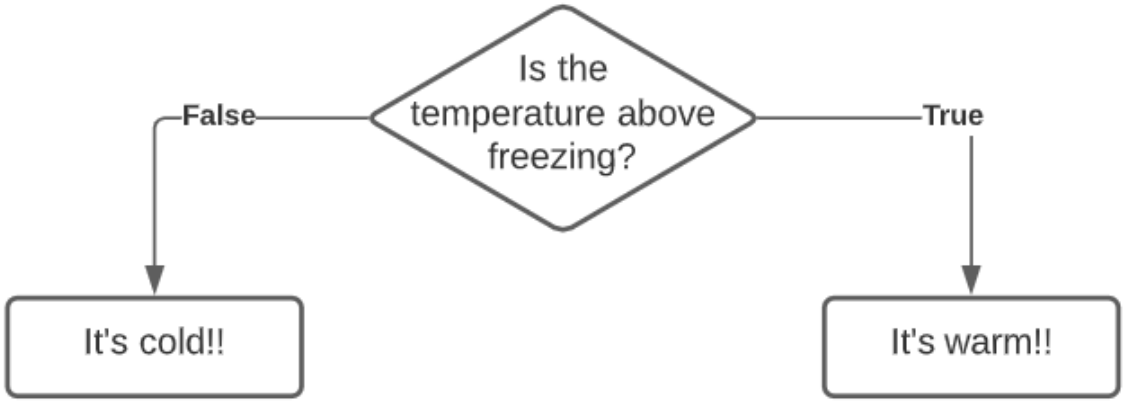
```
if answer == 'Y':

if (answer == 'Y'):
```

```
  File "<ipython-input-133-ac0bdbaa2d06>", line 2
    if (answer == 'Y':)
                      ^
SyntaxError: invalid syntax
```

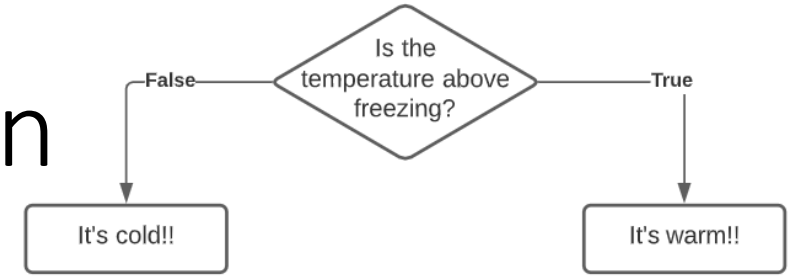
Introduction to Python

Conditional Statements

```
if condition:
    statements
else:
    statements
```



Introduction to Python

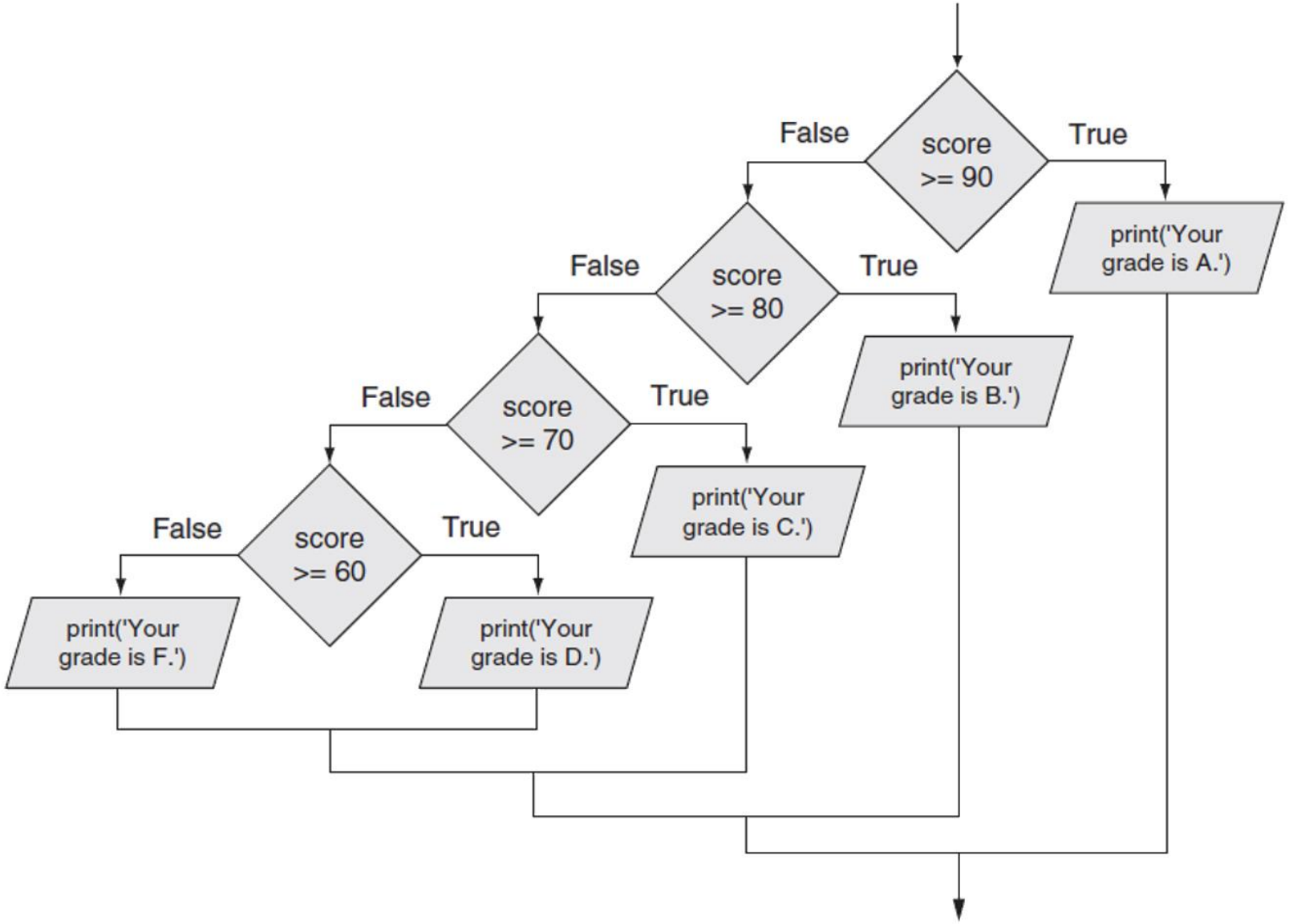


Conditional Statements

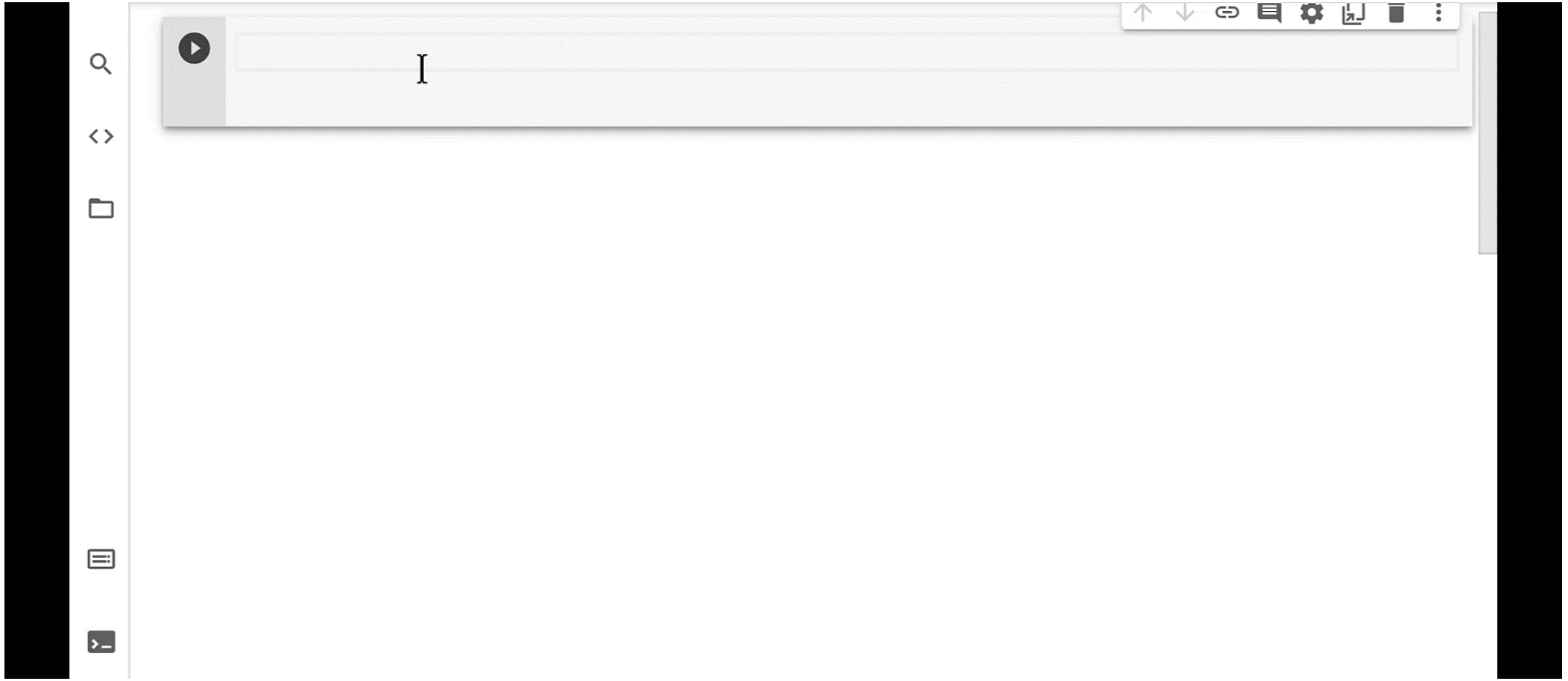
```
answer = input("Is the temperature above freezing (Y/N) ")
if answer == 'Y':
    print("It's cold!!")
else:
    print("It's warm!!!")
```

Introduction to Python

```
if score >=
    A
else if sc
    B
else if sc
    C
else if sc
    D
else F
```



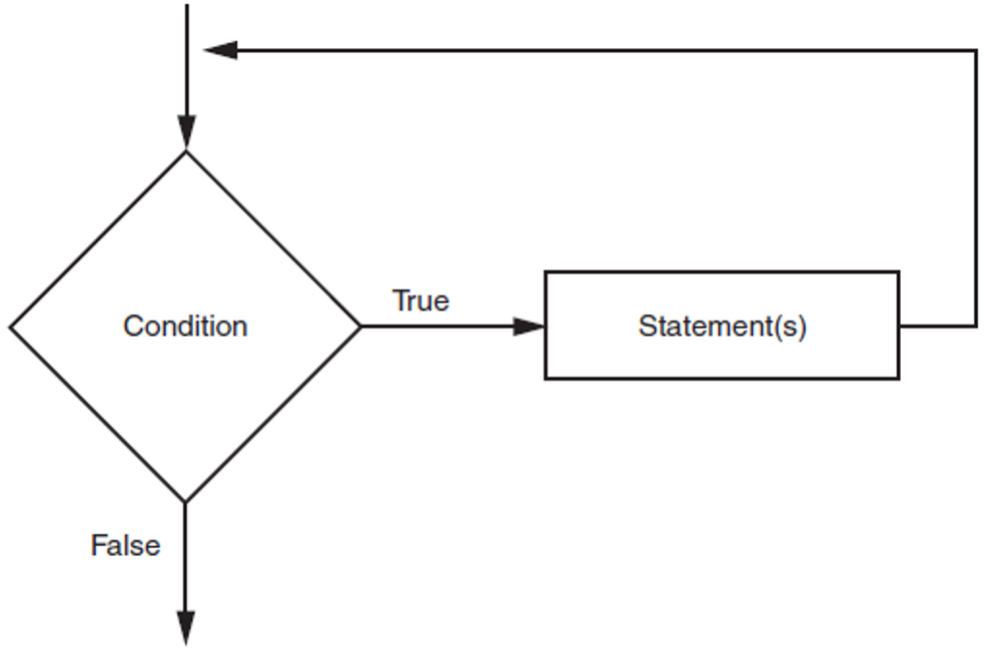
Introduction to Python



Introduction to Python

Condition-Controlled Loop

```
while condition:
    statements
```



Introduction to Python

```
import random
ActualNumber = random.randrange(0,100)

print("Guess My Number")
Guess = int(input('Enter a number between 1 and 100 '))
while (ActualNumber != Guess):
    if Guess < ActualNumber:
        print("Your guess is too low")
    else:
        print("Your guess is too high")
    Guess = int(input('Enter a number between 1 and 100 '))
print("You guessed correctly!!")
```

Introduction to Python

```
import random
ActualNumber = random.randrange(0,100)
```

Python defines a set of functions that are used to generate or manipulate random numbers through the `random` module.

Functions in the random module rely on a pseudo-random number generator function `random()`, which generates a random float number between 0.0 and 1.0. These particular type of functions is used in a lot of games, lotteries, or any application requiring a random number generation.

Introduction to Python

```
import random
ActualNumber = random.randrange(0,100)
```

These particular types of functions are used in a lot of games, lotteries, or any application requiring a random number generation.

```
randrange(beg, end, step)
```

The random module offers a function that can generate random numbers from a specified range and also allowing rooms for steps to be included.

Introduction to Python

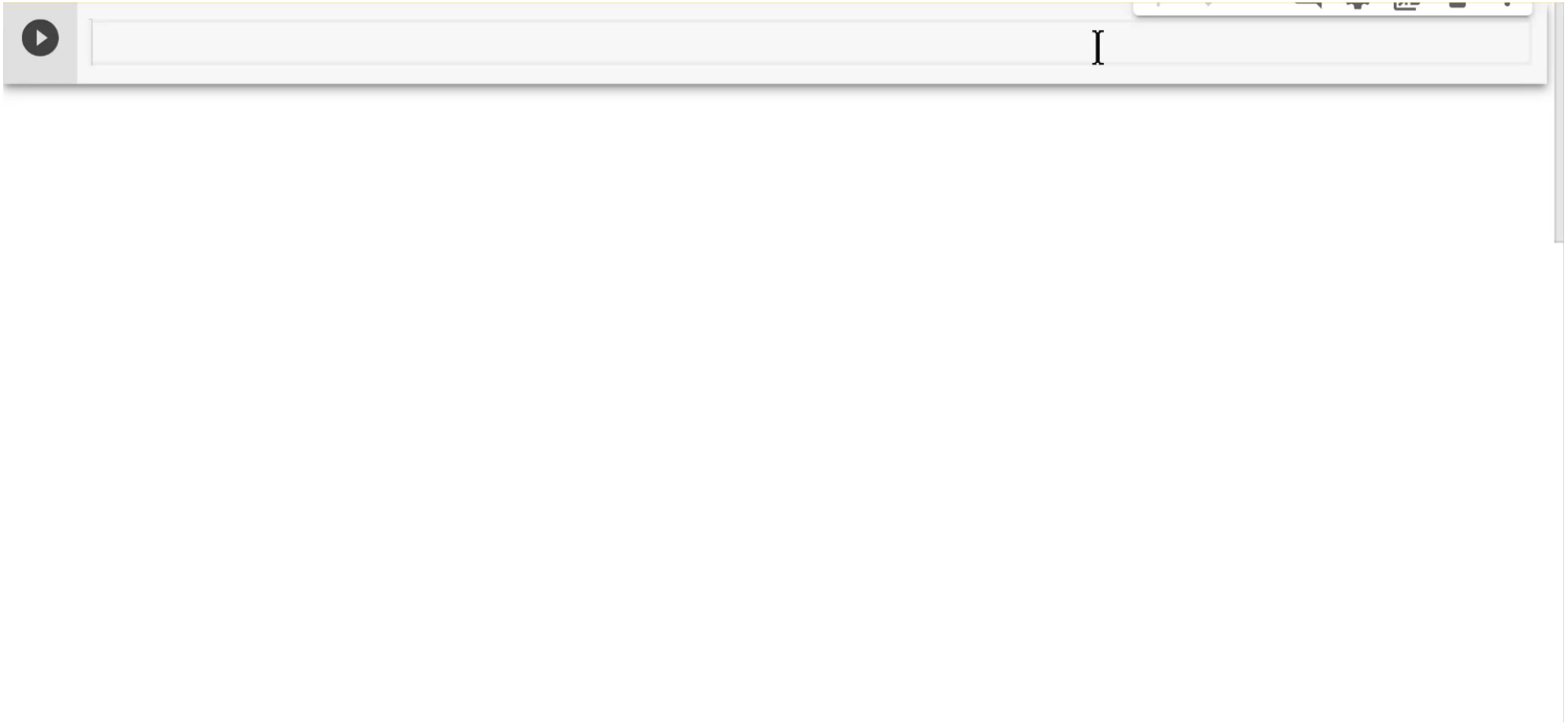
C and Java and C++ all have similar forms of `for` loops

```
for (int i = 0; i < 10; i++)
```

This same logic in Python is a little different

```
for i in range(0,10):
```

Introduction to Python



Introduction to Python

```
for i in range(0,10):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

Introduction to Python

Introduction to Python

```
start = int(input("Enter start "))
end = int(input("Enter end "))
for i in range(start, end):
    print(i)
```

Introduction to Python

```
counter = 0
Word = input("Enter a string ")
LetterToCount = input("Enter a letter ")
for iterator in Word:
    if iterator == LetterToCount:
        counter+=1
print("{0:s} contains {1:d} instances of '{2:s}'"
      .format(Word,counter,LetterToCount))
```

Introduction to Python

`list`

A `list` is just like dynamic sized array

Think of a `vector` in C++ and `ArrayList` in Java.

A `list` in Python is not required to be homogeneous

a `list` can contain different data types

A `list` is mutable

they can be altered after creation (strings are not mutable)

Introduction to Python

`list`

A `list` is ordered and the number of elements can be counted.

Elements in a `list` are indexed starting with 0

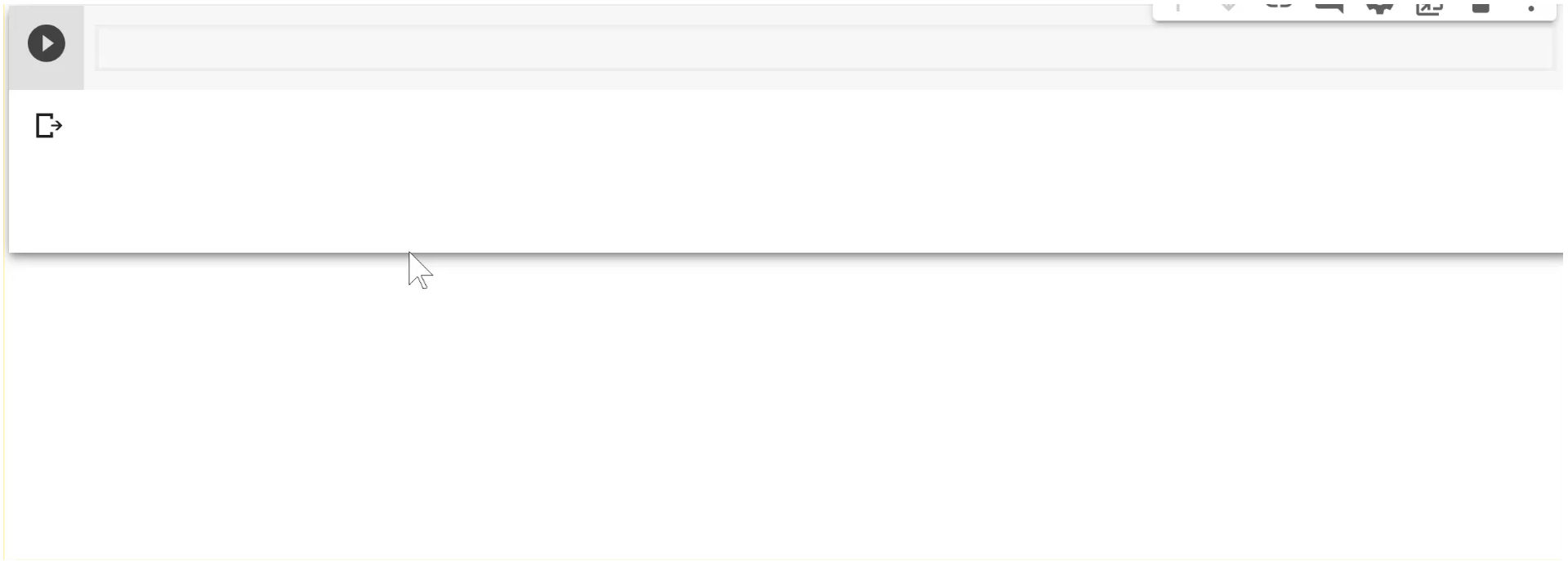
`list` elements are accessed using `[]`

Many of the things you know about arrays, vectors and ArrayLists can be used with `list` in Python.

Introduction to Python

`list`

How is a `list` created?



Introduction to Python

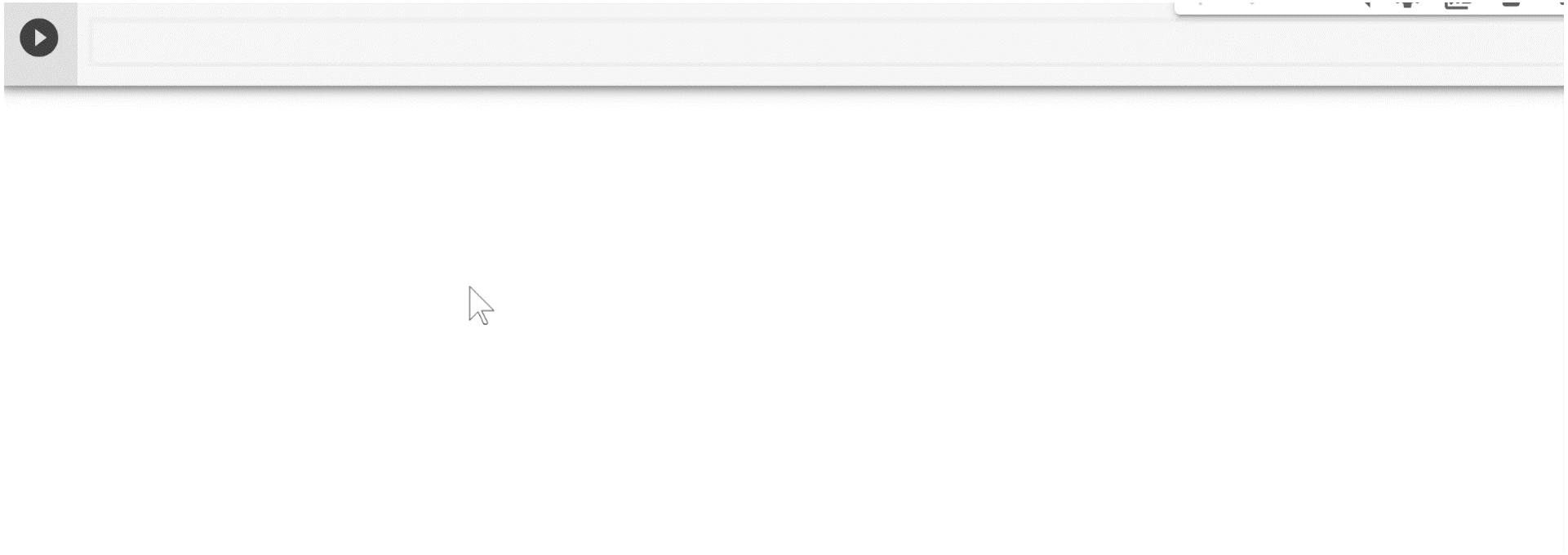
Indexing a `list` in Python has an interesting "twist"

```
ShoppingList = ['Apples', 'Butter', 'Carrots', 'Dragon Fruit']
print(ShoppingList[0])
print(ShoppingList[-2])
print(ShoppingList[len(ShoppingList)-1])
print(ShoppingList[-3])
```

```
Apples
Carrots
Dragon Fruit
Butter
```

Introduction to Python

`list`



+ can be used to concatenate lists

Introduction to Python

```
MyList = [1,2,3]
YourList = ['a','b','c']
TheirList = [1.2, 3.4, 5.6]
OurList = MyList + YourList + TheirList
print(OurList)
print(MyList)
print(YourList)
print(TheirList)
```

```
[1, 2, 3, 'a', 'b', 'c', 1.2, 3.4, 5.6]
[1, 2, 3]
['a', 'b', 'c']
[1.2, 3.4, 5.6]
```

Introduction to Python

```
MyList = [1,2,3]
YourList = ['a','b','c']
TheirList = [1.2, 3.4, 5.6]
OurList = MyList*3 + YourList + TheirList
print(OurList)
print(MyList)
print(YourList)
print(TheirList)
```

```
[1, 2, 3, 1, 2, 3, 1, 2, 3, 'a', 'b', 'c', 1.2, 3.4, 5.6]
[1, 2, 3]
['a', 'b', 'c']
[1.2, 3.4, 5.6]
```

Introduction to Python

Let's take a slice of our `list` (slice – a span of items that are taken from a sequence).

```
MyList = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]
```

Introduction to Python

```
MyList = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]

print(MyList[0:9])
print(MyList[2:9])
print(MyList[0:])
print(MyList[:15])
print(MyList[:len(MyList)])
print(MyList[2:15:2])
print(MyList[2:15:3])
print(MyList[15:2:-3])
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8]

[2, 3, 4, 5, 6, 7, 8]

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]

[2, 4, 6, 8, 10, 12, 14]

[2, 5, 8, 11, 14]

[15, 12, 9, 6, 3]
```

Introduction to Python

`list[start:end]`

A slice is a `list` containing copies of elements from *`start`* up to, but not including, *`end`*

If `start` is not specified, `0` is used for start index

If `end` is not specified, then `len(list)` is used for end index

Slicing expressions can include a step value and negative indexes relative to end of `list`

Introduction to Python

How to determine if an item is contained in a `list`

```
Animal = input("Enter an animal ")
MySafari = ['Elephant', 'Giraffe', 'Lion', 'Tiger']
if Animal in MySafari:
    print("It's over there!!")
else:
    print("I don't see one...")
```

`in` can be used to determine if a given input is a member of a `list`. Returns `True` if the item is in the `list` and returns `False` if the item is not in the `list`.

Introduction to Python

How to add an item to a list

```
Alphabet = []
for i in range(0,26):
    Alphabet.append(i)
print(Alphabet)
```

`list.append(item)`

adds items to a `list`

`item` is appended to the end of the existing `list`

Introduction to Python

How to determine where an item is located in a list

`list.index(item)`

used to determine where an item is located in a `list` - returns the index of the first element in the list containing `item`

Introduction to Python

`list.insert(`*`index, item`*`)`

used to insert *`item`* at position *`index`* in the list

`list.sort()`

used to sort the elements of the list in ascending order

`list.remove(`*`item`*`)`

removes the first occurrence of *`item`* in the list

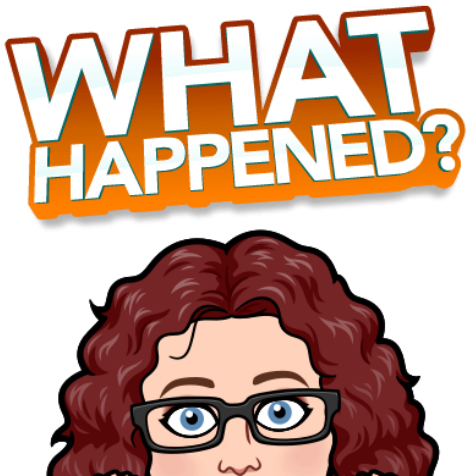
`list.reverse()`

reverses the order of the elements in the list

Introduction to Python

How do we make a copy of a `list`?

```
List1 = [1,2,3,4]
List2 = List1
print(List1)
print(List2)
```



Introduction to Python

So how do we make an actual copy of a list and not just create another variable that points to the same address?

```
List1 = [1,2,3,4]
List2 = []
List2 += List1
print(List1)
print(List2)
List1.remove(2)
print(List1)
print(List2)
```

```
[1, 2, 3, 4]
[1, 2, 3, 4]
[1, 3, 4]
[1, 2, 3, 4]
```

Introduction to Python

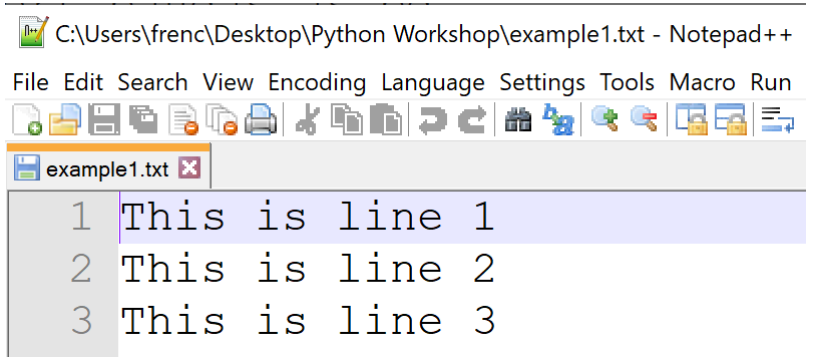
```
List1 = [1,2,3,4]
List2 = []
for n in range(0, len(List1)):
    List2.append(List1[n])
print(List1)
print(List2)
```

```
[1, 2, 3, 4]
[1, 2, 3, 4]
```

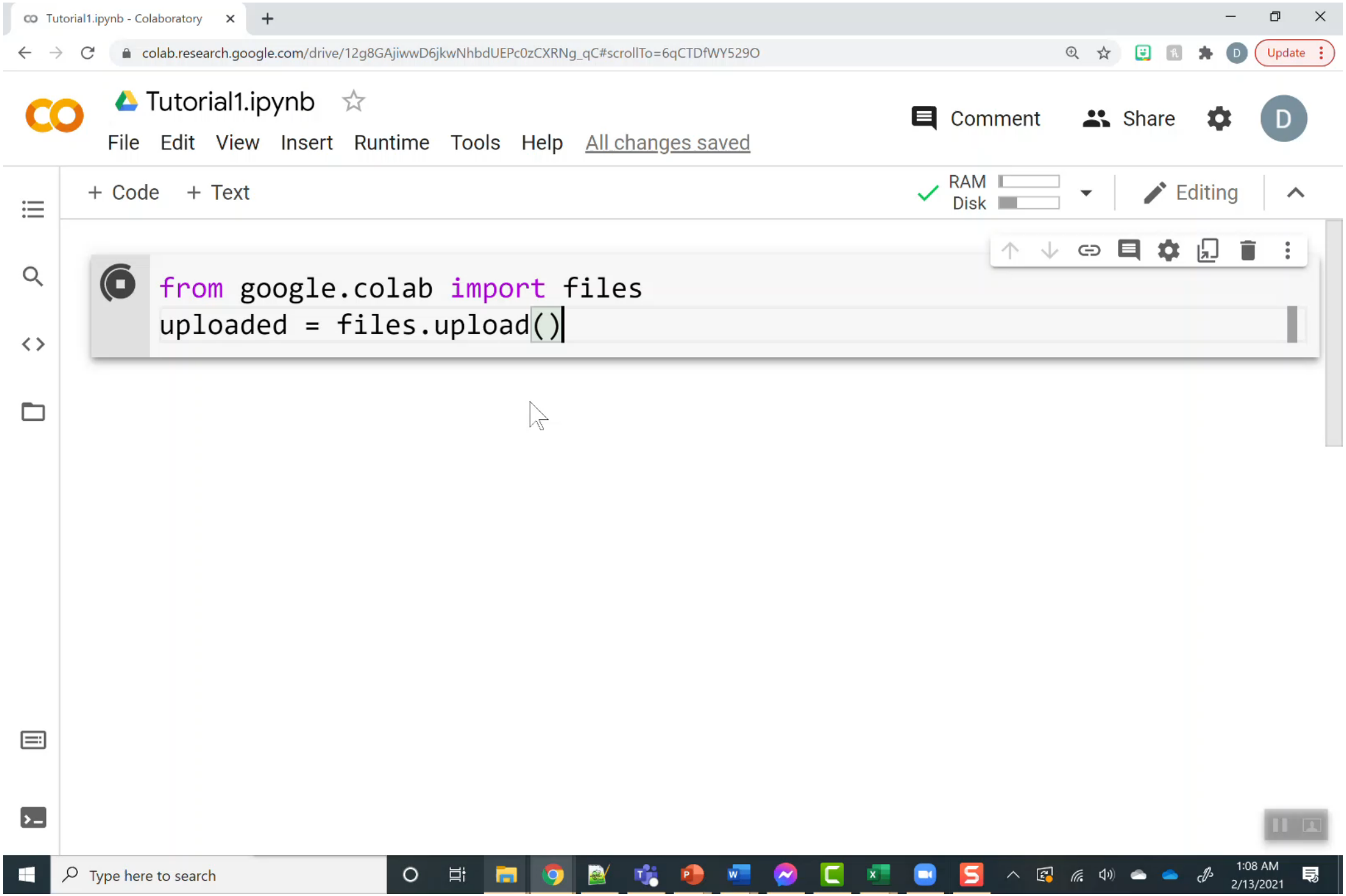
Introduction to Python

Now we are going to learn how to read and write files.

Let's start off with a sample file.



We can create the file on our own computer and then upload it into Colab with some code.


Tutorial1.ipynb - Colaboratory
colab.research.google.com/drive/12g8GAjiwwD6jkwNhbdUEPc0zCXRNg_qC#scrollTo=6qCTDfWY529O
Update
Tutorial1.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
D
+ Code + Text
RAM
Disk
Editing
from google.colab import files
uploaded = files.upload()
Type here to search
1:08 AM
2/13/2021

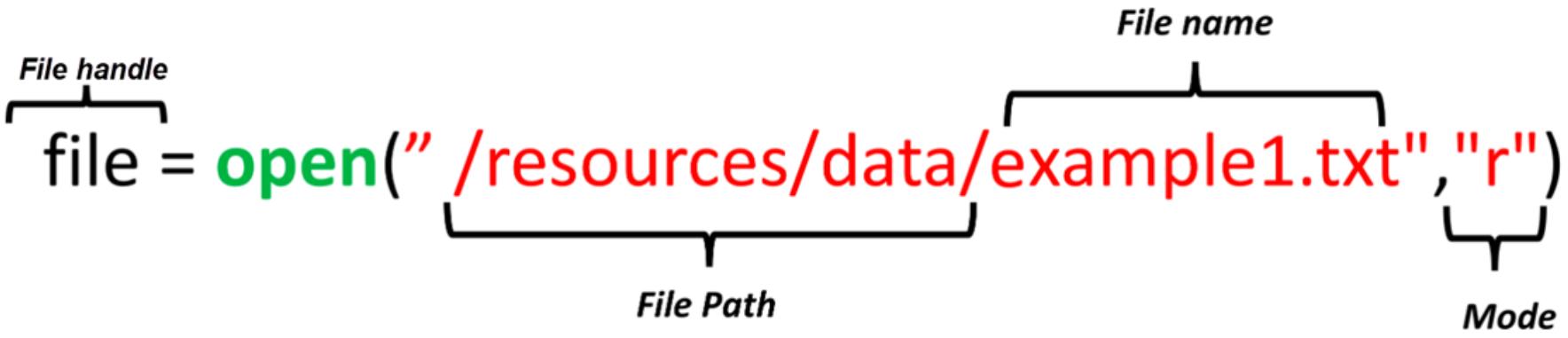
Introduction to Python

The path to our file, `example1.txt`, within the Colab Notebook is

`/content/`

Use `!pwd` to verify

One way to open a file is by using `open()` with a file handle.



Introduction to Python

```
MyFile = open("/content/example1.txt", "r")
FileContent = MyFile.read()
print(FileContent)
print(len(FileContent))
MyFile.close()
print(MyFile.closed)
```

```
This is line 1
This is line 2
This is line 3
45
True
```

Introduction to Python

A Better Way to Open a File

Using the `with` statement is better practice.

It automatically closes the file even if the code encounters an exception.

The code will run everything in the indent block then close the file object.

Introduction to Python

```
with open("/content/example1.txt") as MyFile:
    FileContent = MyFile.read()
    print(FileContent)
    print(len(FileContent))
    print(MyFile.closed)
```

+ Code + Text

The default mode is "r"

```
This is line 1
This is line 2
This is line 3
```

Introduction to Python

We are not required to read the whole file into a string at once. We can read certain amounts of characters.

```
with open("/content/example1.txt") as MyFile:
    print(MyFile.read(5))
    print(MyFile.read(3))
    print(MyFile.read(5))
    print(MyFile.read(1))
```

```
This
is
line
1
```

Introduction to Python

```
This is line 1
This is line 2
This is line 3
```

```
with open("/content/example1.txt") as MyFile:
    print(MyFile.read(2))
    print(MyFile.read(1))
    print(MyFile.read(4))
    print(MyFile.read(7))
```

```
Th
i
s is
 line 1
```

Introduction to Python

```
This is line 1
This is line 2
This is line 3
```

```
with open("/content/example1.txt") as MyFile:
    print(MyFile.read(2))
    print(MyFile.read(1))
    print(MyFile.read(4))
    print(MyFile.read(70))
```

```
Th
i
s is
 line 1
This is line 2
This is line 3
```

Introduction to Python

We can also read the whole line at once

```
with open("/content/example1.txt") as MyFile:
    print(MyFile.readline())
```

```
This is line 1
```

Introduction to Python

A `for` loop can read each line in the file until it reaches the end of the file.

```
with open("/content/example1.txt") as MyFile:
    for Line in MyFile:
        print(Line)
```

```
This is line 1

This is line 2

This is line 3
```

Introduction to Python

Code can be added to act upon each file line

```
with open("/content/example1.txt") as MyFile:
    i = 65
    for Line in MyFile:
        print("{0:c}. {1:s}".format(i, Line))
        i+=1
```

```
A. This is line 1

B. This is line 2

C. This is line 3
```

Introduction to Python

Each line of the file can be added to a `list` as an element

```
with open("/content/example1.txt") as MyFile:
    MyFileList = MyFile.readlines()
    print(MyFileList[0])
    print(MyFileList[1])
    print(MyFileList[2])
```

```
This is line 1

This is line 2

This is line 3
```

Introduction to Python

Each line from the file is an element in the `list`

```
with open("/content/example1.txt") as MyFile:
    MyFileList = MyFile.readlines()
    print(MyFileList[2])
    print(MyFileList[1])
    print(MyFileList[0])
```

```
This is line 3
This is line 2

This is line 1
```

Introduction to Python

Let's write to a file

```
with open("/content/example2.txt", 'w') as MyFile:
    MyFile.write("This is line A")
```

How can we tell if this worked? We could open the file for read and print it.

```
with open("/content/example2.txt", 'r') as MyFile:
    print(MyFile.read())
```

```
This is line A
```

Introduction to Python

There is another way to verify…

```
!ls -l examp*
with open("/content/example2.txt", 'w') as MyFile:
    MyFile.write("This is line A")
!ls -l examp*
!more example2.txt

-rw-r--r-- 1 root root 45 Dec  5  2018 example1.txt
-rw-r--r-- 1 root root 45 Dec  5  2018 example1.txt
-rw-r--r-- 1 root root 14 Feb 13 06:11 example2.txt
This is line A
```

Introduction to Python

```
with open("/content/example2.txt", 'w') as MyFile:
    MyFile.write("This is line A")
    MyFile.write("This is line B")
!more example2.txt
```

Introduction to Python

```
with open("/content/example2.txt", 'w') as MyFile:
    MyFile.write("This is line A\n")
    MyFile.write("This is line B")
!more example2.txt
```

Introduction to Python

We can write a `list` to a file.

```
ShoppingList = ["Bread", "Peanut Butter", "Jelly"]
with open("/content/example3.txt", 'w') as MyFile:
    for item in ShoppingList:
        MyFile.write(item)
!more example3.txt
```

Introduction to Python

We can copy one file to another

```
!more example3.txt
with open("/content/example3.txt", 'r') as ReadFile:
    with open("/content/example4.txt", 'w') as WriteFile:
        for FileLine in ReadFile:
            WriteFile.write(FileLine)
!more example4.txt
```

```
1. Bread
2. Peanut Butter
3. Jelly
1. Bread
2. Peanut Butter
3. Jelly
```


Tech Support
Tech support,
How may I help you?
8:53 AM
Hi,
I've got a problem. Your program is telling me to get a pet snake. I don't want one.
8:53 AM
Tech Support
Excuse me?
8:54 AM
It's giving me a message telling me I need a snake to run it.
8:54 AM
Tech Support
Read the message to me please.
8:55 AM
Error: Python required to run the script.
8:55 AM

Introduction to Python